МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

_____ /Вадим ГОРЯЧКІН/

«_20_» _____12_____ 2021 р.

ДИПЛОМНА РОБОТА

на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Дослідження стохастичних та стохастико-детермінованих алгоритмів сортування**

Theme **Research of stochastic and stochastic-determined sorting algorithms**

Керівник дипломної роботи проф. _____ Віктор ШИНКАРЕНКО

Нормоконтролер доц. _____ Олена КУРОП'ЯТНИК

Студент групи ПЗ2021 _____ Костянтин ГАЛАНІН

Student Kostiantyn HALANIN

Дніпро – 2021

Дніпровський національний університет залізничного транспорту
імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем Кафедра Комп'ютерні інформаційні технології

Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ проф. Шинкаренко В.І.
(підпис)
«___»_____ 2021 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС магістр
(освітній ступінь)

студента групи (ПЗ2021) 951-М Галаніна Костянгина Костянтиновича
(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження стохастичних та стохастико-детермінованих алгоритмів сортування. Затверджена наказом по університету від «18» листопада 2020 р. № 690ст.
«24» червня 2021 р. №345ст.

ом закінченої роботи «21» грудня 2021 р.

3 Вихідні дані до дипломної роботи _____
_____
_____
_____

4 Зміст пояснювальної записки (перелік питань до розробки) Призначення, постановка задачі та огляд аналогів, методика проведення дослідження, проектування і розробка інструментального програмного забезпечення, дослідження швидкодії алгоритмів сортування, охорона праці та безпека в надзвичайних ситуаціях, висновки.

5 Перелік демонстраційного матеріалу Постановка задачі, опис аналогів, методи вирішення, механізми реалізації, аналіз результатів, висновки.

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	доц. Гненний М. В.	11.11.21	14.11.21
Охорона праці та безпека в надзвичайних ситуаціях	проф. Саблін О. І..		

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва розділів дипломного проекту	Термін виконання розділів проекту (роботи)	Примітка
1	Вступ	1.09.2020 – 10.09.2021	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	10.09.2020 – 15.09.2021	
3	Постановка задачі, технічне завдання	15.09.2020 – 30.09.2021	
4	Створення тестової програми	01.10.2020 – 15.10.2021	
5	Перші тестування	15.10.2020 – 22.10.2021	30%
6	Аналіз результатів	30.10.2020 – 11.11.2021	
7	Розрахунок економічних показників	11.11.2020 – 14.11.2021	
8	Охорона праці	14.11.2020 – 18.11.2021	60%
9	Оформлення пояснювальної записки	18.11.2020 – 01.12.2021	100%
10	Демонстраційні матеріали	5.11.2020 – 16.12.2021	

Дата видачі завдання «18» листопада 2020 р.

Керівник дипломного проекту _____ (підпис) В.І. Шинкаренко (ПІБ)

Завдання прийняв до виконання _____ (підпис) К.К. Галанін (ПІБ)

РЕФЕРАТ

Об’єкт дослідження: стохастичні та стохастико-детерміновані алгоритми сортування

Предмет дослідження: часова ефективність стохастичних та стохастико-детермінованих алгоритмів сортування.

Мета кваліфікаційної роботи: підвищення ефективності алгоритмів сортування використовуючи стохастику.

Методи дослідження та апаратура: для вирішення поставлених задач використовувався персональний комп’ютер, розроблене ПО та математичні методи статистики для оцінки результатів. Експерименти проводили евристичним методом.

Результати та їх новизна: новизна полягає у тому, що вперше було корисно використано стохастику в алгоритмах сортування. В результаті виконання кваліфікаційної роботи було значно пришвидшено існуючі алгоритми сортування і визнано користь використання стохастики в алгоритмах сортування.

Пояснювальна записка складається зі вступу, 5 частин, 6 додатків:

- у вступі описується сутність розробки. Складається з 3 сторінок;
- у першому розділі наведено аналіз сучасного стану проблеми дослідження. Складається з 14 сторінок;
- у другому розділі надано обґрунтування експериментального методу дослідження. Складається з 6 сторінок;
- у третьому розділі представлено проектування й розробка інструментального забезпечення для дослідження. Складається з 4 сторінок;
- у четвертому розділі описано виконані дослідження. Складається з 21 сторінки;
- у п’ятому розділі розглянуто охорону праці. Складається з 10 сторінок.
- додатки містять технічне завдання та робочий проект.

Рисунків – 25, таблиць – 9, джерел – 58.

Перелік ключових слів: алгоритм, сортування, стохастика, швидкодія, стохастико-детермінований алгоритм сортування.

ЗМІСТ

ВСТУП

Актуальність дослідження: сучасному світі одним з найважливіших ресурсів є інформація. Інформація генерується та використовується усюди. Та кожну годину інформації стає все більше і більше. Обсяг інформації зростає в геометричній послідовності.

Згідно з дослідженням [39] за 2020 рік, на одну людину за секунду створюється 1,7 мегабайт інформації. В день компанією Facebook генерується приблизно 4 петабайта інформації щодня. Загалом за минулий рік користувачі мережі інтернет створили 51 зеттабайт даних.

На сьогодні персональні сайти та інформаційні системи є майже у кожному бізнесі, і постає питання у обробці та збереженні великої кількості інформації. Так, в часи комп'ютеризації усіх процесів постає питання ефективного сортування великих масивів даних.

З кожним роком створюються нові алгоритми сортування, покращуються існуючі, комбінуються і усе з однією метою, зробити час виконання певного алгоритму менше тим самим зробивши його використання більш ефективним.

Зараз відомо десятки алгоритмів які мають назву, що читається, та сотні, якщо не тисячі алгоритмів з номерними назвами, або просто невідомі нам. Усі вони детерміновані і заточені під окремі задачі. Та якщо вхідний масив даних буде сильно відрізнятись від бажаних умов, алгоритм буде виконуватись найгірший з можливих проміжок часу, займати процесор, та в підсумку гальмувати бізнес процеси пов'язані з ним.

Згідно з дослідженням аналогів, при спробах покращити існуючи алгоритми сортування, або розробити новий, люди не зверталися до стохастики.

Тема роботи: «Дослідження стохастичних та стохастико-детермінованих алгоритмів сортування»

Об'єкт дослідження: процес алгоритму сортування.

Предмет дослідження: часова ефективність стохастичних та стохастико-детермінованих алгоритмів сортування.

Мета роботи полягає у визначенні ефективності застосування стохастичних та стохастико-детермінованих алгоритмів сортування в сучасних комп'ютерних системах.

Поставлена мета зумовила необхідність вирішення такого комплексу взаємопов'язаних завдань:

- реалізувати стохастичний алгоритм сортування;
- реалізувати інструментальне програмне середовище для оцінки часу виконання алгоритму за заданими показниками;
- дослідити стохастичний алгоритм сортування;
- дослідити використання стохастичного алгоритму сортування разом з детермінованими алгоритмами;
- визначити ефективність використання стохастичного та стохастико-детермінованих алгоритмів у порівнянні з існуючими, та визначити доцільність їх використання.

Методи дослідження: для дослідження часової характеристики алгоритмів сортування було побудовано окреме програмне забезпечення. Дослідження стохастичного алгоритму виконувалось методом спроб та помилок, а саме коефіцієнти поступово змінювались та досліджувались зміни для пошуку оптимального. При оцінюванні результатів заміру часових характеристик алгоритмів було зроблено по 100 сортувань, пораховано час виконання, та за допомогою методів математичної статистики приведено середні показники для кожної ситуації, та порівняні з іншими, для подальшої оцінки їх ефективності.

Наукова новизна: вперше було використано стохастику в алгоритмах сортування, завдяки чому значно зменшено час виконання окремих алгоритмів.

Практичне призначення: результати виконаних досліджень явно вказують на доцільність використання стохастики разом з алгоритмами сортування для покращення їх швидкодії, що покращує їх ефективність.

Під час дослідження було розроблено декілька стохастико-детермінованих алгоритмів сортування, що працюють краще за існуючі зараз детерміновані алгоритми сортування. Їх впровадження в сучасні системи дозволить виконувати сортування великих об’ємів даних швидше, що дозволить раціонально використовувати процесорний час та час роботи комп’ютерної системи.

Апробація результатів дослідження: основні положення магістерської роботи доповідалися та були схвалені на вісімдесят першій всеукраїнській науково-технічній конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту» (Дніпро, ДНУЗТ, 28 жовтня 2021 року).

Публікації за темою роботи: за результатами роботи опубліковано одну наукову працю, а саме тези доповіді на всеукраїнській конференції [40].

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

АС – алгоритмічна структура

ВД – вхідні дані

ДА – детермінований алгоритм

ЕРМ - електронна розрахункова машина

КС – комп’ютерна система

ОС – операційна система

ПЗ – програмне забезпечення

ПЗС – програмні засоби

ПК – персональній комп’ютер

ПП – псевдовипадкова послідовність

РСА – розрахункова складність алгоритму

СА – стохастичний алгоритм

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМИ ШВИДКОДІЇ ІСНУЮЧИХ ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ ТА ВИКОРИСТАННЯ СТОХАСТИЧНИХ АЛГОРИТМІВ СОРТУВАННЯ

1.1 Аналіз предметної сфери

1.1.1 Опис проблеми. Актуальність досліджень.

В епоху комп’ютеризації уся друкована інформація переводиться у цифровий вигляд. Кожна людина стає окремим генератором інформації. З кожним роком генерується все більше і більше нової інформації, що наведена на рисунку 1.1[1].

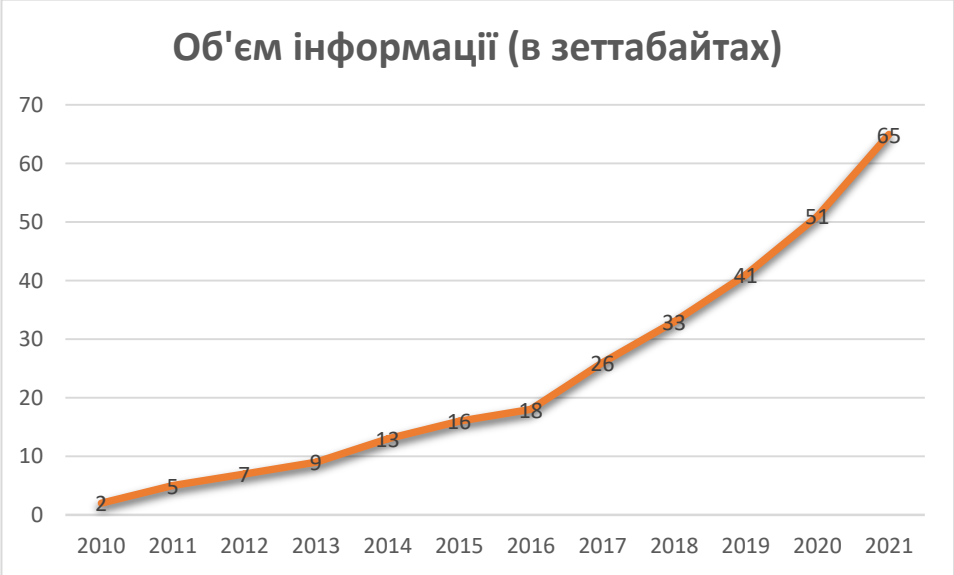

Рисунок 1.1 — Кількість інформації у світі

Зі збільшення інформації цифрових носіях постає питання з її оптимального сортування. Збільшення кількості інформації потребує більше ресурсів комп’ютерної системи та часу.

Люди постійно порівнюють різні існуючі алгоритми сортування на швидкодію, вигадують нові алгоритми, але усі відомі алгоритми не використовують стохастику.

Суть дослідження полягає у перевірці гіпотези, що стохастичних та стохастико-детерміновані алгоритми можуть показати кращі результати, та працювати значно легше і швидше.

У разі підтвердження гіпотези буде виведено найоптимальніший варіант алгоритму сортування.

1.2 Огляд аналогів серед детермінованих алгоритмів сортування

Обробка інформації безпосередньо використовує різні алгоритми сортування, написані на різних мовах, та на різних комп'ютерних системах. Алгоритми використовуються різними способами та з різними цілями, що значно впливає на швидкодію обробки в цілому.

1.2.1 Сортування міхуром

Сортування міхуром, або сортування простим обміном – це найпростіший алгоритм сортування для реалізації та розуміння. Є ефективним тільки на невеликих масивах даних.

У чистому вигляді майже не використовується, але використовується в основі деяких інших алгоритмів сортування, а саме у швидкому, пірамідальному та в шейкерному алгоритмах сортування.

Сам алгоритм сортування складається з повторюваних проходів по усьому масиву. За кожен прохід порівнюється порядок двох сусідніх елементів, і якщо порядок невірний, елементи переставляються. Масив вважається відсортованим, коли за прохід не було зроблено ні однієї перестановки елементів.

Кожен прохід здвигає найбільший елемент у кінець рядка, на своє місце, поряд з іншими найбільшими елементами, а найменші переміщаються до початку масиву. Звідси і назва міхур, найбільший елемент спливає.

1.2.2 Швидке сортування

Один з найшвидших алгоритмів сортування. Універсальний. Є покращеним алгоритмом прямого обміну, представником якого є сортування міхуром, що має низьку ефективність.

Суть алгоритму, що перестановки відбуваються на максимальній відстані, після чого масив поділяється на дві частини.

Під час сортування виконується три етапи:

- обирається опорний елемент;
- масив поділяється на дві частини, так що елементи менші за опорне число стають з одного боку, а більші – з іншого. Якщо число дорівнює опорному, воно стає з більшими;

– виконуються усі ці етапи до отриманих масивів, якщо кількість елементів масиву більша за один.

Опорний елемент обирається з середини алгоритму. Є інші алгоритми отримання опорного елементу, коли опорний елемент обирався з початку масиву, з кінця, що знижувало ефективність на частково відсортованих алгоритмах, та отримання медіани з першого, середнього та останнього елементів, але це ускладнює алгоритм, тому не використовується зазвичай у сортуванні.

1.2.3 Сортування вставкою

На вхід алгоритму подається масив даних, з якого береться по одному елементу, які називаються ключами. Вихідна послідовність спочатку не має елементів.

На кожному кроці обирається один елемент з вхідного масиву та ставиться у правильну позицію у вихідній послідовності. Ці дії повторюються, доки вхідний масив не буде вичерпано.

У будь який проміжок часу вихідний масив буде завжди відсортовано.

1.2.4 Сортування злиттям

Цей алгоритм упорядковує списки.

Алгоритм складається з 3 етапів:

– вхідний масив ділиться навпіл;
– кожна частина сортується окремо, можна одним і тим же алгоритмом;
– отримані відсортовані масиви складаються в один.

Коли використовується даний алгоритм рекурсивно, частини масиви діляться поки розмір їх не стане дорівнювати одиниці. Тому що масив з одного елемента є відсортованим.

На кожному кроці береться найменший елемент з перших двох елементів розбитих масивів та записується до результату. Використаний елемент далі не враховується. Елементи додаються так, поки усі не будуть в результуючому масиві.

Якщо елементи одного з масивів вже перенесені, другий масив автоматично переноситься повністю в кінець результуючого масиву.

1.2.5 Сортування вибором

В стандартній реалізації є нестійким.

Алгоритм складається з 3 етапів:

- визначається номер мінімального елементу масиву;
- виконується заміна мінімального елемента з першим невідсортованим, якщо перший невідсортований не є мінімальним;
- відсортовані елементи виключаються з пошуку мінімальних та невідсортованих. Повторюються усі кроки для наступних елементів масиву.

1.2.6 Сортування Тіма (Tim sort)

Це гібридний алгоритм сортування, який включає в себе сортування вставками та сортування злиттям. Головна ідея в тому, що часто в реальному житті дані частково упорядковані. На таких даних алгоритм показує себе значно ефективніше за багато інших алгоритмів сортування.

Сам алгоритм складається з наступних етапів:

- обчислення мінімального розміру упорядкованої послідовності

Обчислюється на основі розміру вхідного масиву. Має бути не дуже великим, для швидшої роботи сортування вставками.

Має бути не дуже малим, що б зменшити кількість зливань підмасивів. Рекомендується робити розмір рівний степені числа 2, бо сортування злиттям краще працює з рівними по розміру масивами.

- ділення на підмасиви та їх сортування

Складається з наступних етапів:

- вказівник поточного елементу стає в начало вхідного масиву;
- починаючи з поточного елементу починається пошук упорядкованого масиву. Поточний та наступний елемент однозначну в нього входять. Якщо вони стоять навпаки, вони переставляються;

- якщо розмір упорядкованої послідовності менший за мінімальний розмір упорядкованої послідовності, обираються наступні елементи, доки вони не порівняються.

- отриманий підмасив сортується сортуванням вставками. Через невеликий розмір та наявність упорядкованих елементів сортування відбудеться швидко і ефективно.
- показник поточного елементу стає на наступний за підмасивом елемент;
- якщо кінця масиву не достигли, повторюється з другого шагу. Інакше кінець даного етапу.
- злиття;

Злиття відбувається наступним чином:

- створюється пустий стек пар індексу начала підмасива та його розміру. Обирається перший упорядкований підмасив;
- добавляється у стек інформація про поточний підмасив;
- з’ясовується, чи потрібно виконувати злиття поточного підмасиву з попередніми за встановленими правилами;
- якщо одне з правил порушене, попередній масив зливається з меншим. Виконується до тих пір, поки правила не виконаються, або не буде упорядковано усі дані;
- якщо злито не усі підмасиви – береться наступний та виконуються усі кроки починаючи з другого.

Процедура злиття:

- створюється тимчасовий масив розміром з найменший із тих, що зливаємо;
- копіюється найменший в створений;
- обираються перші елементи більшого та тимчасового масивів;

- кожен крок порівнюємо елементи та обираємо найменший та копіюємо в новий відсортований масив. В масиві, з якого був обраний елемент обираємо наступний, та повторюємо цей крок, доки один з масивів не кінчиться;
- елементи які залишились додаються в кінець масиву.

1.2.7 Плавне сортування(Smooth sort)

Алгоритм сортування вибором. Має перевагу при частково упорядкованих даних.

Суть алгоритму схожий на пірамідальне сортування. В масив складається куча даних, які потім сортуються через видалення максимуму з кучі. Використовується куча, яка отримана за допомогою чисел Леонардо. Це дає перевагу, коли вхідний масив частково упорядкований, через пришвидшення створення та видалення куч буде використано значно менше часу.

1.2.8 Сортування Шелла

Основою алгоритму є сортування вставками.

Відмінність у тому, що алгоритм порівнює не тільки елементи, що знаходяться поруч, а і на конкретній відстані. Спочатку порівнюються елементи які стоять на деякій відстані. Потім відстань зменшується, доки не стане дорівнювати одиниці.

Плюси: не потребує пам’яті під стек, не деградує при невдалих даних

Мінуси: в більшості випадків є більш ефективні варіанти

1.3 Стохастика

Стохастика — це випадковість [3]. Випадковий процес, поведінка якого не є детермінованою, та майбутній стан такої системи представляється величинами, які або можуть бути передбаченими або повністю випадковими [19 - 23].

1.3.1 Стохастичні алгоритми

Стохастичні алгоритми – це алгоритми, у основі яких є випадковість. Вони не мають гарантії ні часу виконання, ні якості.

1.3.2 Лінійний стохастичний алгоритм шифрування

Одним з найперспективніших напрямків забезпечення безпеки інформації є використання стохастичних алгоритмів. А саме використання методів захисту інформації прямо або опосередковано основаних на використанні псевдовипадкових послідовностей.

Для цього зараз використовують або стохастичні алгоритми, створені на основі використання еліптичних кривих, або дихотомічні генератори ПП.

Методи СА створених на основі еліптичних кривих є найбільш математично обґрунтованими, але маючи інформацію про принцип її роботи і послідовність виходу постає питання про існування односторонньої функції [11],[12]. Не маючи ключової інформації крипто аналітик може використовувати тільки жереб, для отримання інформації про попередній елемент послідовності.

У цьому випадку, для повного приховання ймовірнісних зв’язків в роботі [10] було досліджено метод шифрування, який дозволяє перетворити повідомлення в послідовність некорелюваних значень. Це лінійний СА шифрування, оснований на канонічному розкладі випадкових послідовностей.

1.3.3 Стохастичні алгоритми вирішення зворотних задач

Зворотні задачі – це визначення приросту аргументів прямої функції [13]. Теорія зворотних задач побудована на основі цільового припису людини у вигляді її приросту та додаткової інформації: начальних значень аргументів та функції та коефіцієнтів відносної важливості [17].

Простота даного засобу та актуальність задач, що він вирішує обумовлюють його використання у різних сферах: економіці, освіті [14 - 16]. На його основі був побудований модифікований метод [18], який при комп’ютерній реалізації не вимагає виконання перевірки відповідності поставлених коефіцієнтів важливості поставленої мети.

При кількості аргументів більше двох задача стає набагато складнішою та виникає потреба виконати згортку.

Використання стохастичних алгоритмів при вирішенні задач такого роду дозволяє уникнути складних обчислень та знайти приблизне рішення беручи до уваги коефіцієнт важливості, обмежень аргументів та навіть коли змінні приймають або тільки цілі, або значення з заданого діапазону.

Отже на допомогу приходить алгоритм випадкового пошуку. Його ідея полягає в обранні випадкових значень аргументів з обраного інтервалу, підрахуванню цільової функції та порівнянні її величини з найкращими з розрахованих. Якщо нове значення менше, то воно запам’ятовується як рішення.

Для функції з одним аргументом послідовність буде наступна:

- генерується величина на заданому інтервалі
- якщо $f(x) < f_{min}$, то нова точка запам’ятовується в якості нового рішення.

Ці кроки повторюються доки не буде виконано або встановлена кількість повторювань, або не буде знайдено рішення з встановленою точністю.

1.4 Огляд літератури

1.4.1 Порівняння складності детермінованих алгоритмів

Під час дослідження алгоритмів не є доцільним виконувати усі дослідження з абсолютного нуля. Усі алгоритми сортування, що хоча б мають назву вже були дослідженні не один десяток разів

Усі теоретичні дані отримано та наведено зручно у вигляді таблиць [2].

Надалі, при дослідженні, спираючись на задачу розроблюваного алгоритму можна усі дані брати готовими та використовувати для того що б обрати алгоритми, які більше підходять для вирішення задачі, або для покращення за певною умовою, яку не складно відслідкувати у приведених таблицях. Таблицю складності алгоритмів, які використовуються найчастіше, як в чистому вигляді, так і у сукупності інших алгоритмів, наведено на наступному рисунку 1.2.

Таблиця 1.1 — Показники складності алгоритмів сортування

Алгоритм	Структура данных	Временная сложность		
		Лучшее	В среднем	В худшем
Быстрая сортировка	Массив	O(n log(n))	O(n log(n))	O(n^2)
Сортировка слиянием	Массив	O(n log(n))	O(n log(n))	O(n log(n))
Пирамидальная сортировка	Массив	O(n log(n))	O(n log(n))	O(n log(n))
Пузырьковая сортировка	Массив	O(n)	O(n^2)	O(n^2)
Сортировка вставками	Массив	O(n)	O(n^2)	O(n^2)
Сортировка выбором	Массив	O(n^2)	O(n^2)	O(n^2)
Блочная сортировка	Массив	O(n+k)	O(n+k)	O(n^2)
Поразрядная сортировка	Массив	O(nk)	O(nk)	O(nk)

Наглядно продемонстровано кольорами найкращі та найгірші результати роботи алгоритмів. Зеленим кольором показано ті результати, які є найкращими, жовтим – які є добрими, і доцільними при використанні, та фіолетовим – ті ситуації, у випадку отримання яких алгоритм показує найгірші результати та не рекомендується до використання.

Реальні ж показники часу та складності необхідно отримувати евристично, через те, що в кожній системі та з кожними даними алгоритм буде показувати різні результати.

1.4.2 Порівняння часу виконання алгоритмів

У сучасних дослідженнях порівнюють існуючі алгоритми сортування у пошуку найкращого за обраними характеристиками. В більшості випадків за складовою часу. Найшвидшою вважають швидке сортування, сортування Шелла або сортування Тіма, що показано на рисунку 1.2. І кожне нове дослідження або підтверджує це, або спростовує. Але усі ці дослідження допомагають нам краще зрозуміти принципи роботи алгоритмів та знайти спосіб їх покращення.

Деякі алгоритми сортування значно краще працюють з частково відсортованими даними, що наведено на рисунку 1.3 та 1.4 [4]. Вони були створені саме для роботи з такими даними, але ми не можемо заздалегідь знати, чи будуть дані відсортовані, і у випадку повністю мішаних даних алгоритм показує себе значно гірше.

Саме ці алгоритми є початковою ціллю для дослідження використання стохастики при попередній обробці даних. Тим самим у всіх випадках дані будуть частково відсортовані і алгоритми будуть показувати значно кращі результати за найшвидші на сьогодні.

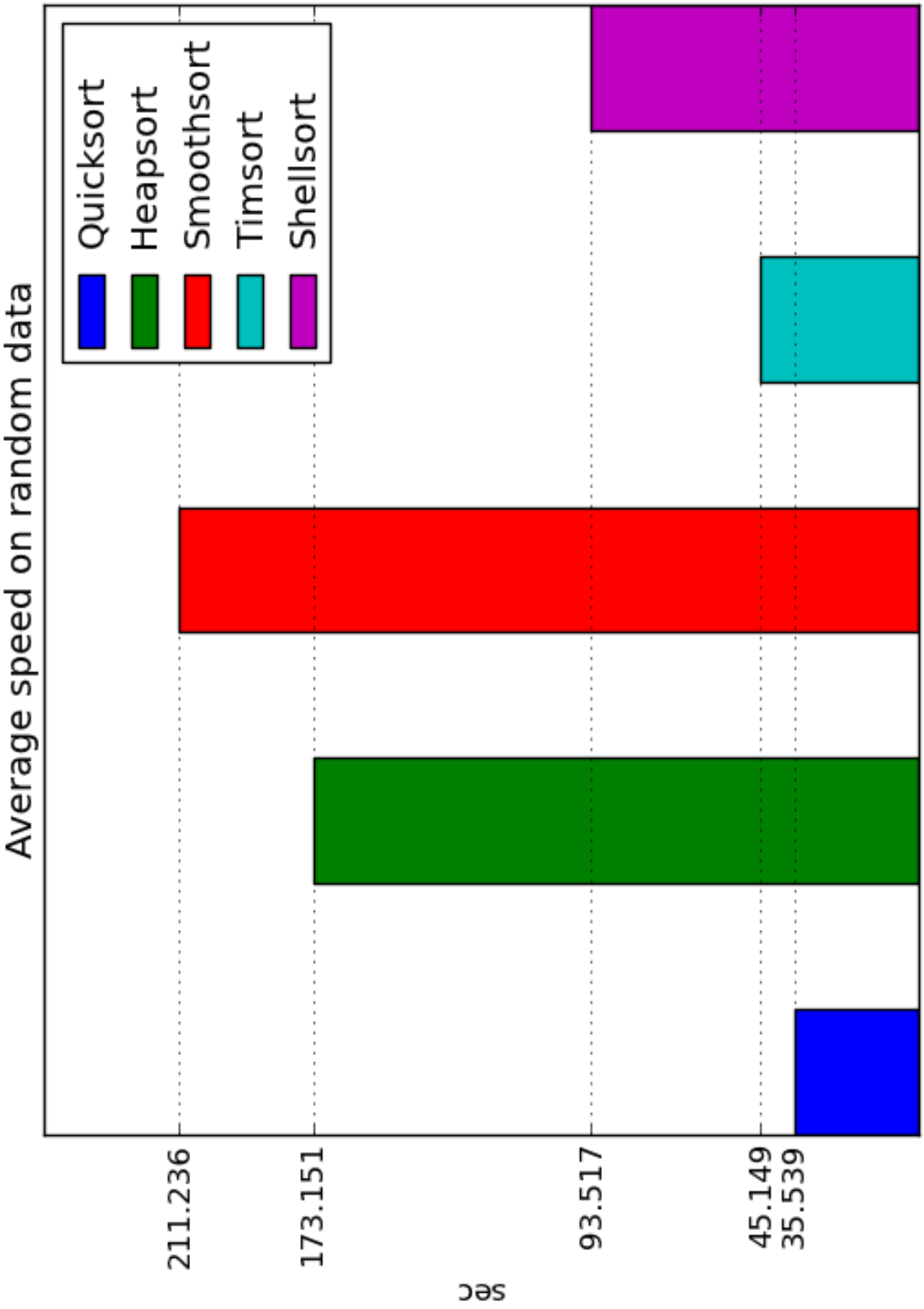

Рисунок 1.2 — Порівняння алгоритмів сортування на випадкових даних

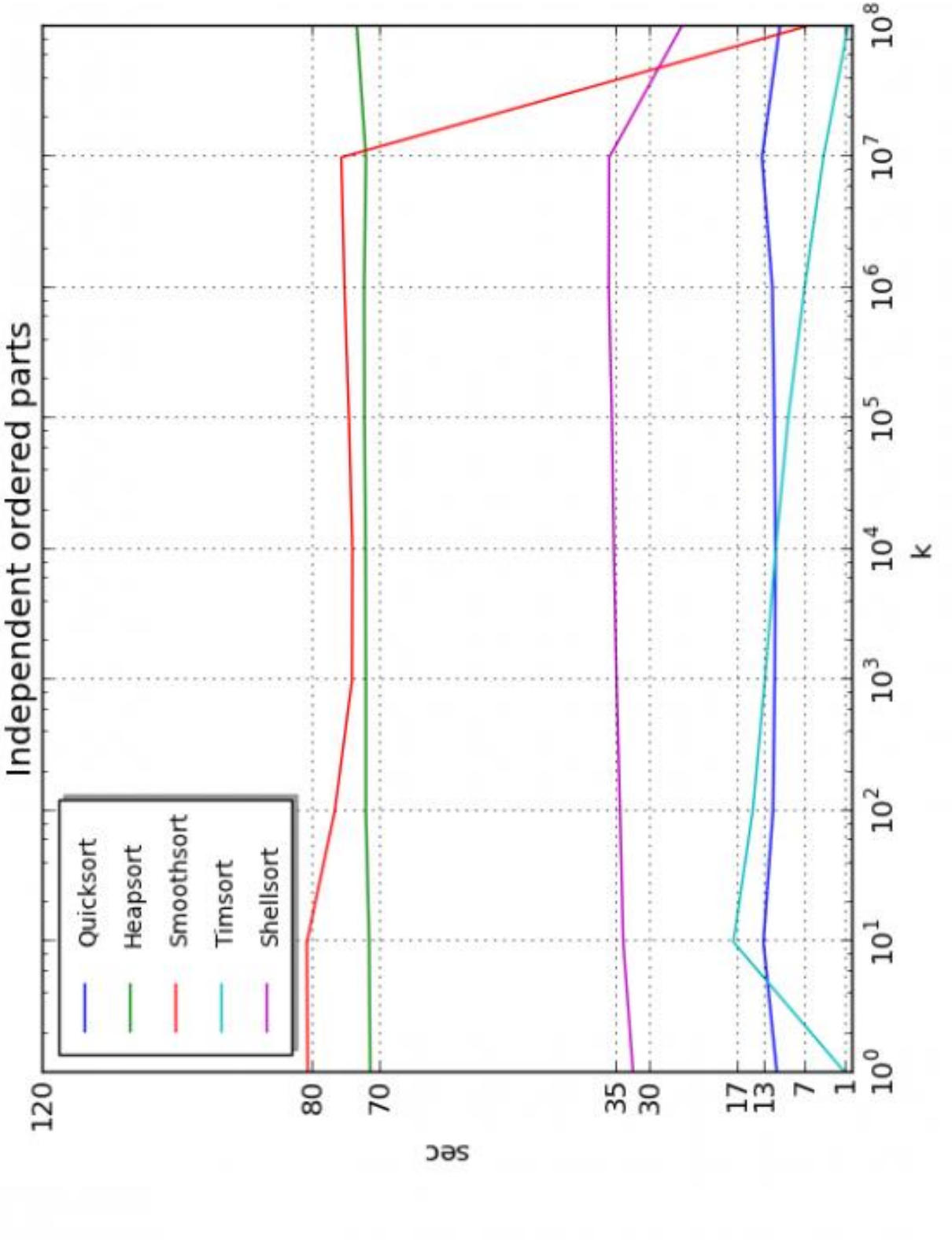

Рисунок 1.3 — Сортування відсортованих масивів перемішаних по частинам

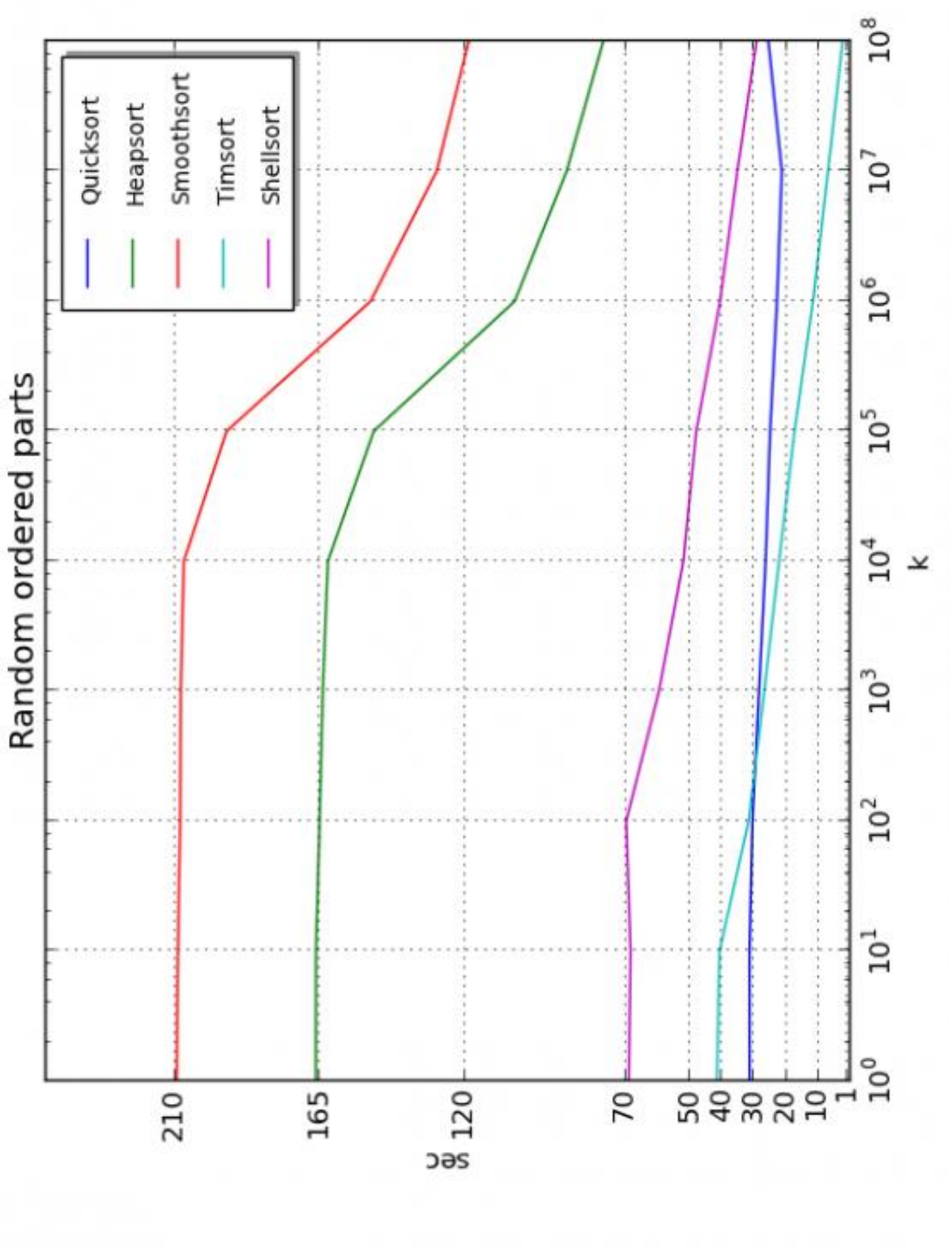

Рисунок 1.4 — Сортування частково відсортованих масивів даних

1.5 Призначення та сфера застосування

Розроблений програмний засіб призначений для виконання сортування обраним алгоритмом, та отримання часових характеристик одного з реалізованих алгоритмів для масивів даних різних розмірів.

Експлуатаційне призначення додатку полягає в отриманні часових даних роботи конкретного алгоритму для подальшого його дослідження, для визначення найефективнішого алгоритму за показниками часу та кількості операцій.

Сфера застосування. Додаток може використовувати будь хто персонально. Додавати нові алгоритми та сценарії їх використання може користувач, який володіє базовими навичками програмування на мові Java будь якої версії.

1.6 Постановка задачі

Задача розробити програмній засіб для дослідження часових характеристик наступних алгоритмів сортування:

- стохастичного сортування;
- сортування міхуром;
- швидкого сортування;
- сортування вставкою;
- сортування злиттям;
- сортування вибором;
- сортування Тіма;
- плавне сортування;
- сортування Шелла;
- стохастичного з використанням одного з детермінованих алгоритмів.

Висновки до розділу 1

Згідно дослідження предметної сфери бачимо, що є велика потреба в більш швидких і гнучких алгоритмах сортування, а наразі існуючи алгоритми дуже обмежені. Вирішити цю проблему може використання стохастики для розробки нового алгоритму та покращення існуючих. Отже завдання з розробки програмного засобу є актуальним.

РОЗДІЛ 2.ОБГРУНТУВАННЯ ЕКСПЕРЕМЕНТАЛЬНОГО МЕТОДУ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СТОХАСТИЧНИХ ТА СТОХАИСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Розробка нових більш ефективних алгоритмів сортування на сьогодні особо актуальна. Кількість оброблюваної інформації зростає що секунди, і на вирішення задач потрібно все більше ресурсів та часу КС.

Головною метою експерименту постає підтвердити або відхилити гіпотезу про користь використання стохастики у сортуванні.

2.1 Гіпотеза щодо корисності використання стохастики у сортуванні

Відомі алгоритми сортування є детерміновані і не включають в себе елементів стохастики. Згідно з цим ставимо гіпотезу, що використовуючи стохастичні та стохастико-детерміновані алгоритми можна пришвидшити виконання сортування, та вирішувати більш різноманітні задачі через їх гнучкість. А саме:

- використовуючи стохастичний алгоритм сортування можна виконувати приблизне сортування;
- використання стохастичного алгоритму сортування разом з детермінованими значно пришвидшує роботу детермінованих алгоритмів;
- використання стохастичного алгоритму сортування разом з детермінованим дає кращі результати ніж використання тільки детермінованих алгоритмів.

2.2 Часова складність алгоритму

При порівнянні алгоритмів одним з найважливіших показників є часова складність алгоритму.

Часова складність алгоритму, або обчислювана складність описує час, який буде затрачено на використання алгоритму. Для визначення якої рахуються елементарні операції , які виконує алгоритм під час роботи. Зазвичай з результатів обирається найгірший результат отриманий на даних одного розміру, і позначається як максимальний час виконання алгоритму при заданому розміру даних.

Але при роботі зі стохастикою однозначно неможливо визначити складність СА, через те, що алгоритм може виконатись як один так і безліч разів. Саме тому в цьому дослідженні часовою складністю алгоритму буда вважатись конкретний час виконання алгоритму на одній КС з конкретним розміром вхідних даних.

2.3 Розрахункова складність алгоритму

Для визначення ступеню зростання розрахункової складності алгоритму (далі РСА) при зростанні об’єму вхідних даних(далі ВД) використовують асимптотичні показники РСА. Насамперед, це показник точної асимптотичної оцінки, який включає в себе верхню та нижню асимптотичну оцінки.

Асимптотичні оцінки РСА найчастіше, але не завжди співпадають з асимптотичними оцінками часу виконання алгоритмів. Прискорення розрахунків завдяки їх розпаралелювання при конвеєрній обробці команд виконаний на декількох процесорах не залежить від об’єму ВД, які оброблюються.

Показники асимптотичної РСА можна вважати метриками з порядковою шкалою. По ній алгоритми розподіляються на класи, які визначають потенційний час виконання алгоритмів при значних об’ємах даних. Приклади практичного використання асимптотичних показників наведені в [27, 28].

Найбільш корисною є класифікація за якою алгоритми розподілені на P алгоритми, а саме розрахунки по ним обмежені поліноміальною асимптотикою, та NP – алгоритми, розрахунки яких не обмежуються поліноміальною асимптотикою [29 - 34].

Асимптотичні показники РСА розкривають потенційні можливості відносно кількості розрахунків та часу виконання алгоритмів при визначених умовах, а саме: значного збільшення об’єму даних, зменшенню похибок розрахунки і так далі. Такий розподіл на класи дуже корисний для визначення потенційної цінності алгоритму потенційні можливості його використання в програмному забезпеченні (далі ПЗ)

Для вирішення задачі вибору алгоритму цей показник може бути корисним, коли потреби конкретного програмного засобу (далі ПЗС) дійсно пов’язані з вимогами до об’єму ВД.

Якщо є декілька алгоритмів одного класу за асимптотичним показником РСА, вибір за показником стає неможливим, але він дозволяє звузити коло найбільш підходящих алгоритмів.

Але для задачі оцінки алгоритму який тільки що розробили це мало що значить, а для задачі вибору алгоритму при близьких асимптотиках також, враховуючи перспективу невідомого розпаралелювання розрахунків при їх виконанні на сучасних електронних розрахункових машинах (далі ЕРМ). Для багатьох алгоритмів асимптотичні показники є дуже близькими, а різниця показників навіть у декілька разів може бути нівельована різним ступнем розпаралелювання розрахунків та кешуванням.

Подивимось на ситуацію в [45]. Деякий процесор має спеціалізований функціональний пристрій для виконання якихось розрахунків, який розраховує, наприклад, суму елементів паралельно послідовно.

Якщо в алгоритмі є цикл для розрахунку вказаної суми, то його розрахунок має розрахункову складність $\Theta(n)$. Уявимо, що транслятор розпізнав такий цикл та перетворив його у відповідну команду процесору, та процесор виконує лиш одну операцію. Таким чином завдяки транслятору та розрахунковому пристрою РСА знижена до $\Theta(1)$.

Тобто окремий алгоритм та програма на різних розрахункових пристроях можуть мати різну розрахункову складність.

Кількість паралельних операцій в спеціалізованому функціональному пристрої завжди обмежена фізичними можливостями. Якщо об’єм даних буде перевищувати можливості процесора, заміна циклу однією операцією стане неможливою, і збільшення об’єму ВД все ж приведе до розрахункової складності $\Theta(n)$.

Але може відбутися і так, що об’єм даних при усіх можливих застосуваннях дозволить вказану вище заміну. Тобто розрахункова складність в практичному використанні буде $\Theta(1)$.

Підводячи висновки розуміємо, що залежність асимптотичних показників РСА залежить від ЕРЗ.

2.3.1 Залежність розрахункової складності алгоритмів від програмно-апаратних серед перетворення та функціонування алгоритмів

Залежно від «апаратури» Д. Кнутом [36, 37] зроблені наступні припущення:

- усі команди виконуються однаковий час;
- час виконання команд не залежить від даних.

Друге припущення неявне та виходить з першого.

Розглянемо друге припущення. Усі числа при обчислення на ЕРМ моделюються деякими модельними числами. Цілі числа моделюються в 2-, 4-, 8-байтовими числами за або без знаку. Розрядність числа не залежить від його значення. Числа 14 та 2400, наприклад будуть представлені однаково 16- бітними (або 2- байтовими) модельними числами. Час виконання конкретної операції для чисел, представлених однаковими модельними числами, буде однаковий.

Але уявімо, що значення чисел є надзвичайно великими та моделюються 16-байтовими числами. Якщо ЕРМ не має команд обробки таких чисел, то розрахунки відбудуться з двома 8- байтовими числами декількома операціями. Збільшення значень приведе до моделювання 24-, 32- … -байтовими числами та відповідно до збільшення кількості операцій. Тобто чим більше модельні числа, тим більше часи виконуються операції над ними, а значить, і час виконання алгоритму.

Це підтверджує, що тезис, що обчислювана складність алгоритму залежить від даних в цілому, а не тільки від їх кількості та об’єму.

З іншої сторони якщо ЕРМ має команди процесора з 8-, 16-, 32- … -байтовими модельними числами, РСА залишається незмінною при відповідному зростанню значень чисел.

Уявимо, що операції над розрядами чисел виконуються не як на сучасних ЕРМ паралельно-послідовно, а сугубо послідовно, як це виконується власноруч. Однією операцією буде операція над одним розрядом. Тоді кількість операцій буде залежати від розрядності елементів та розрахункова складність значно збільшиться.

Приведені аргументи вказують на те, що кількість обчислень по алгоритму, а саме РСА залежить від виконавчого пристрою.

Враховуючи вищесказане, можна говорити о неявних операціях алгоритмів, котрі виконуються, або можуть виконуватися виконавчим пристроєм та впливають на РСА.

Таким чином РСА залежить від способу виконання алгоритму та при її визначенні слід враховувати ці особливості.

Як відомо [38] алгоритми мають бути масовими. Для алгоритмів сортування це означає використання їх з різними даними метричних просторів.

Однак ефективність алгоритмів для різних типів даних може суттєво змінюватись. Це відбувається через різний час як виконання базових операцій, так і операцій доступу до них.

Архітектура EPM розвивається по шляху збільшення продуктивності комп'ютерів за рахунок розпаралелювання процесу обчислень. Збільшуються технічні можливості розпаралелювання. А саме:

- багатопроцесорність
- MMX технологія, яка реалізує підхід «одна команда – багато даних», наприклад однією командою підсумовується шість пар чисел
- конвеєрна обробка команд – декілька команд виконуються одночасно на конвеєрі в різних стадіях.

Як впливають подібні нововведення архітектури EPM на оцінку ефективності алгоритмів?

Перші два підходи, а саме технологія MMX та багатопроцесорність, для покращення продуктивності вимагають реорганізації алгоритмів та спеціальних методів програмування.

2.4 Основа експерименту

Основою експерименту є два поставлених досліди.

2.4.1 Дослідження стохастичного алгоритму сортування

В основі досліду полягає конструювання стохастичного алгоритму та дослідження його кількісних характеристик. А саме:

- за скільки операцій алгоритм відсортує найкраще заданий масив даних;
- яка ефективність операцій, на кожному етапі сортування.

2.4.2 Дослідження стохастико-детермінованого алгоритму сортування

В основі досліду полягає об’єднання стохастичного алгоритму отриманого у першому досліді, та використання його з різними існуючими алгоритмами, та дослідження його характеристик. А саме:

- чи є використання стохастичного алгоритму з детермінованими доцільним;
- з якими алгоритмами він показує себе найкраще, а з якими використовувати не доцільно;
- до якого моменту буде ефективно використовувати стохастичний алгоритм.

Висновки до розділу 2

У розділі визначено основні характеристики за якими будуть досліджуватися алгоритми та наведено план дослідження їх.

РОЗДІЛ 3.ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНСТРУМЕТАЛЬНИХ ЗАСОБІВ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СТОХАСТИЧНИХ ТА СТОХАСТИКО-ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

3.1 Формалізація задачі

Сценарії використання додатку наведені у якості можливостей користувача.

Користувач може виконувати наступні варіанти використання:

- встановити необхідні параметри експерименту;
- запустити тестування алгоритму;
- сортувати заданим алгоритмом свій масив даних.

3.2 Базова архітектура системи

Програма поділяється на три окремі частини пов’язані між собою з розподілом функцій. Мейн (Main) – основна частина в якій визначаються параметри вхідних даних масиву, та виконується запуск експерименту. Здійснюється регулювання кількості експериментів, та визначення середнього часу їх виконання. Менеджер масиву (ArrayManager) виконує усі операції над масивом, над яким ми проводимо експерименти. А саме, тут обирається алгоритм, який буде виконуватись та дані, які будуть виведені в результаті виконання. Менеджер містить у собі усі алгоритми які використовуються для виконання експерименту згідно з завданням. Та Експеримент (Experiment) який містить методи для вибору алгоритму сортування, та створює вхідний масив для проведення дослідження.

3.3 Проектування

3.3.1 Вибір мови програмування

Для розробки було обрано об’єктивно-орієнтовану мову розробки Java.

Java – сучасна об’єктно-орієнтована і типо-захищена мова програмування.

Вона надає мовні конструкції для підтримки такої концепції роботи. Далі наведено основні аспекти, що забезпечують їй надійність:

- лямбда-вирази для функціонального програмування;
- обробка винятків дозволяє швидко і зрозуміло знаходити помилки та одразу їх вирішувати;
- автоматичне збирання сміття;

- асинхронність;
- строга типізація, що зменшує помилки через несумісність типів до мінімуму.

3.3.2 Технологічна платформа

Розробка відбувалась на платформі Intelij IDEA.

Intelij IDEA – це інтегрована середа розробки програмного забезпечення для багатьох мов програмування, серед яких є Java.

3.3.3 Ієрархія та взаємодія класів

Результати проектування класів наведено на рисунку 3.1.

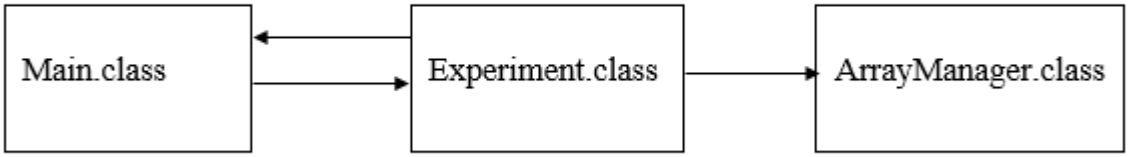

Рисунок 3.1 — Взаємодія класів додатку

3.3.4 Принципи проектування в проекті

При проектуванні було використано 2 принципи SOLID, а саме:

- принцип єдиної відповідальності – для кожного класу повинно бути одне призначення;
- принцип відкритості/закритості – програмні сутності повинні бути відкритими до розширення, але закритими до модифікації [5].

3.3.5 Шаблони проектування в проекті

При проектуванні програмного продукту було використано наступні шаблони:

- шаблон делегування – об’єкт зовні показує деяку поведінку, але насправді передає відповідальність за поведінку іншому пов’язаному об’єкту;
- шаблон функціонального дизайну – гарантує, що кожний окремий модуль має тільки одну відповідальність, та виконує її з мінімумом ефектів на інші частини програми;

- незмінний інтерфейс – створення незмінного об’єкту;
- отримання ресурсу є ініціалізація – Отримання деякого ресурсу поєднано з ініціалізацією, а його звільнення – з видаленням або знищенням об’єкту;
- ланцюжок зобов’язаностей – призначений для організації у системі рівнів відповідальності;
- нуль об’єкт – запобігає передачі значень нуль у системі;
- стан – використовується, коли під час виконання об’єкт має змінювати поведінку враховуючи стан [6].

3.4 Тестування та налагодження програми

Тестування ПЗ – це процес дослідження програмного продукту з наміром перевірити схожість його реальної поведінки на очікувану на кінцевому набору тестів. Цей процес є дуже важливим та трудомістким під час розробки ПЗ.

Найчастіше в сучасній розробці виконується функціональне тестування, та інтеграційне тестування.

3.4.1 Функціональне тестування

Функціональне тестування – це тестування ПЗ в цілях перевірити реалізованість функціональних вимог, а саме здатність ПЗ в конкретних умовах вирішувати задачі, які вони мають вирішувати. Функціональне тестування визначає що саме ПЗ, які задачі може вирішувати [7].

3.4.2 Інтеграційне тестування

Інтеграційне тестування – тестування коли окремі програмні модулі об’єднуються та тестуються групою. В якості вхідних даних використовуються модулі, виконуються тести, визначені у плані тестування, та представляються у якості вихідних даних, призначених для наступного тестування [8].

Ціллю інтеграційних тестів є перевірка відповідності вимогам надійності

Висновки до розділу 3

Згідно з SOLID принципами було виконано проектування інструментального засобу дослідження стохастичних та стохастико-детермінованих алгоритмів.

Було передбачено можливість додавання нових алгоритмів сортування через використання окремого класу на одному рівні з класом Експеримент. Це дозволяє розширювати функціонал розробленого додатку.

Використання шаблонів проектування робить програму легше читаною та більш простою для розуміння. Згідно з цим в майбутньому набагато легше буде її розширювати та виконувати її підтримку

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СТОХАСТИЧНИХ ТА СТОХАСТИКО-ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

4.1 Підготовка до експерименту

Методи планування експериментів дозволяють знизити до мінімуму кількість необхідних випробувань, встановити раціональний порядок та умови проведення досліджень [24 - 26].

Планування експерименту включає в себе наступні етапи:

- встановлення цілі експерименту;
- уточнення умов проведення експерименту;
- виявлення та вибір вхідних даних та параметрів;
- встановлення потрібної точності результатів вимірів;
- складання плану та проведення експерименту;
- статична обробка результатів експерименту;
- пояснення результатів та формулювання рекомендацій згідно їх використання.

4.1.1 Опис використаного програмно апаратного середовища

Дослідження проводилося з використанням персонального комп’ютеру(далі ПК), з встановленною операційною системою Windows 10. Далі наведено важливі характеристики цього ПК.

Процесор

- Intel Core i7-7700U;
- 4 ядра по 2,25 ГГц.;
- Об’єм кеш пам’яті 8 МБ.

Оперативна пам’ять:

- Модель Samsung DDR4L;
- тип DDR4L;
- ємність одного модуля 8 ГБ;
- загальна ємність 16 ГБ.

Відеокарта:

- модель Nvidia GEFORCE 940MX;
- тип пам’яті GDDR5;
- об’єм відеопам’яті 2ГБ.

Пам’ять диску:

- модель Kingston A2000 250 ГБ;
- тип твердотілий накопичувач;
- інтерфейс PCI Express 3.0;
- тип пам’яті GDDR5;

4.1.2 Опис підходу для визначення часу роботи алгоритму

У експерименті буде визначено саме час виконання алгоритму, не враховуючи генерацію масиву, отримання результатів та інші дії, не задіяні в роботі алгоритму.

Визначення час проводилося завдяки вбудованим в мову бібліотекам. Було використано метод nanoTime з класу System.

Так як з невеликими об’ємами алгоритм виконується менш ніж за одну мілісекунду, час рахується в наносекундах, для точного відображення

4.1.3 Вплив кеш-пам’яті на результати вимірювань

Дуже важливо, щоб під час виконання експерименту ні які зовнішні фактори не впливали на результат. Одним із таких факторів є кеш пам’ять. Зберігаючи деякі операції процесор пришвидшує виконання однакових шматків. Тому при повторному виконуванні час виконання буде значно менше. Чим більше кєш пам’яті, тим більше буде різниця. На рис 4.1 наведено таку ситуацію.

На малюнку бачимо, що час виконання однакового алгоритму з одним розміром зменшився майже вдвічі.

Була спроба переповнити кеш пам’ять іншими операціями, числами, навіть іншого типу, але алгоритми починали вести себе хаотично, тому спроби були припинені.

Для чистоти експерименту в результатах будуть наведені дані отримані з великої вибірки, але з ефектом пришвидшення завдяки кеш пам’яті.

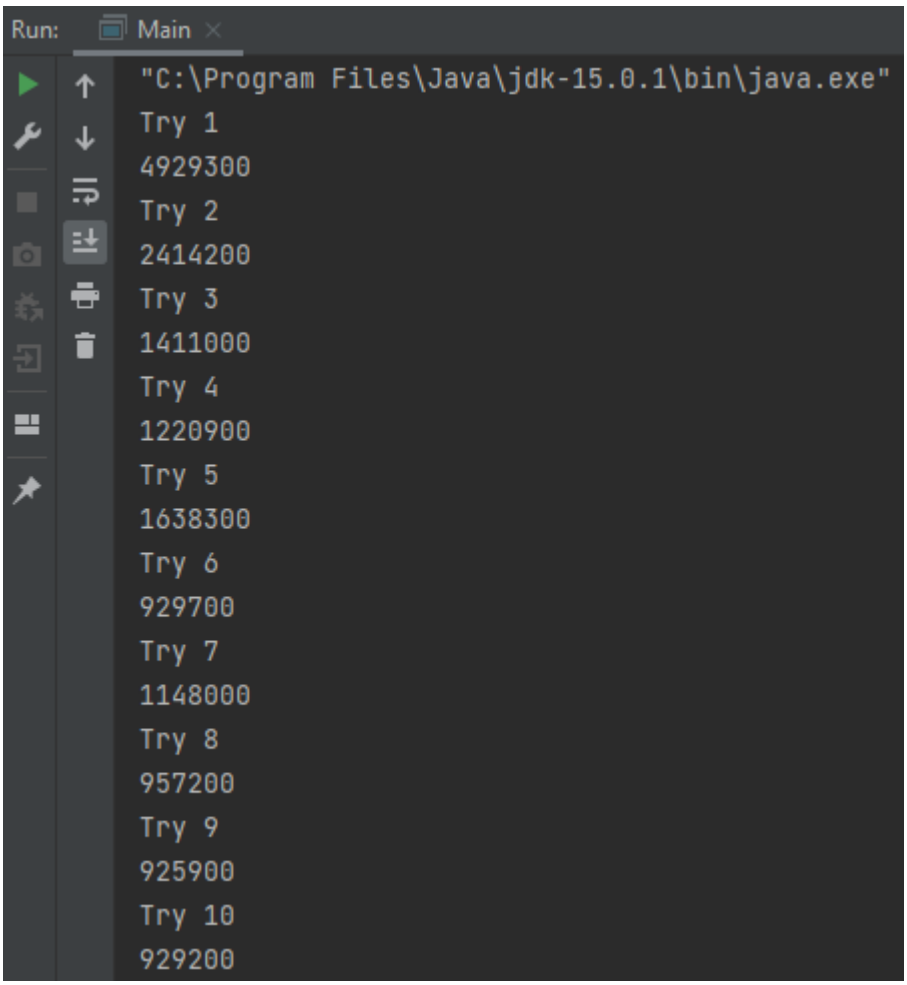

Рисунок 4.1 — Вплив кеш пам'яті на результат експерименту

4.2 Проведення експерименту

4.2.1 Дослідження стохастичних алгоритмів сортування

Гіпотеза була наступна: стохастичний алгоритм сортування може замінити детерміновані алгоритми.

Спочатку генерується випадково масив даних. Дані можуть бути будь якого типу, але для простоти реалізації в програмному додатку реалізовано сортування цілих чисел.

Далі генерується випадкова пара чисел, та якщо ліве більше правого, вони міняються місцями. Так виконується декілька разів і перевіряється алгоритм на відсортованість.

4.2.2 Дослідження стохастико-детермінованих алгоритмів сортування

Гіпотеза була наступна: використання стохастики в детермінованих алгоритмах сортування може покращити час виконання алгоритму, зробивши його більш ефективним.

Цей дослід складається з двох менших.

Перший дослід використання стохастичного алгоритму разом з детермінованим та замір часу. Таким методом ми бачимо ефективність впровадження стохастики.

Другий дослід – пошук ефективного коефіцієнта виконання стохастичного алгоритму для збереження ефективності його використання. Багато ітерацій можуть перекрити всю користь від використання стохастичного алгоритму через велику кількість операцій, які не призвели до сортування масиву.

4.3 Результат експерименту

В результаті проведення першого експерименту було отримано дані, наведені в таблиці 4.1.

Спираючись на отримані дані у порівнянні зі детермінованими алгоритмами стало зрозуміло, стохастичний алгоритм сортування не зможе замінити детерміновані алгоритми через дві причини.

По-перше, він не гарантує часу сортування. Він може виконуватись або годину, або хвилину. Коли буде відсортовано нікому не відомо.

По-друге, він не гарантує високої точності сортування. Відсортувавши масив певну кількість разів ми не можемо гарантувати, скільки відбулося замін.

Таблиця 4.1 — Час приблизного сортування

Кількість елементів масиву	500	1000	5000	10000	50000	100000
Час в секундах	0,003	0,013	0,187	0,702	14,8	57,2
Час в нано-	3135768	13237200	1,88E+08	7,03E+08	1,48E+10	5,72E+10

секундах						

Таке сортування має сенс, коли нам необхідно що б масив був приблизно відсортований. Може бути корисним при створенні задач або прикладів.

В результаті першої частини другого експерименту було отримано дані наведені у таблицях 4.2 - 4.4.

В таблиці 4.2 наведено результати проведення експерименту з масивом в 1000 елементів. Виконується різна кількість ітерацій стохастичного алгоритму спочатку від 100 до 1000, а потім від 10 до 100.

В таблиці 4.3 наведено результати проведення експерименту з масивом в 10000 елементів. Виконується різна кількість ітерацій стохастичного алгоритму спочатку від 1000 до 10000, потім від 100 до 1000, та від 10 до 100.

В таблиці 4.4 наведено результати проведення експерименту з масивом в 100000 елементів. Виконується різна кількість ітерацій стохастичного алгоритму спочатку від 10000 до 100000, потім від 1000 до 10000, від 100 до 1000, та від 10 до 100.

За цими даними необхідно було зрозуміти або розрахувати коефіцієнт кількості ітерацій стохастичного алгоритму при якому його виконання буде найефективнішим.

Згідно таблиць 4.2 - 4.4 були побудовані графіки залежності кількості змін від кількості ітерацій стохастичного алгоритму сортування.

Залежність між отриманими даними відсутня на перший погляд. Але побудувавши графіки наведені на рисунках 4.2 - 4.9 стало зрозуміло, що на досить малих значеннях алгоритм веде себе дуже хаотично, та може не зробити жодної заміни. Коли коефіцієнт високий, алгоритм виконує дуже багато марних дій.

Усі отримані графіки наведені на рисунках 4.13-4.21.

Таблиця 4.2 — Дослідження стохастичного алгоритму сортування на 1000 елементів

	1000									
Кількість елементів	100	200	300	400	500	600	700	800	900	1000
Кількість замін	43	44	33	38	46	39	35	29	43	36
Сумарна кількість замін	43	87	120	158	204	243	278	307	350	386
кількість проходів без дії	57	113	180	242	296	357	422	493	550	614
Кількість елементів	10	20	30	40	50	60	70	80	90	100
Кількість замін	7	6	6	5	5	6	6	6	6	6
Сумарна кількість замін	7	13	19	24	29	35	41	47	53	59
кількість проходів без дії	3	7	11	16	21	25	29	33	37	41

Таблиця 4.3 — Дослідження стохастичного алгоритму сортування на 10 000 елементах

	10000									
Кількість елементів	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Кількість замін	481	489	463	439	418	398	360	381	399	351
Сумарна кількість замін	481	970	1433	1872	2290	2688	3048	3429	3828	4179
кількість проходів без дії	519	1030	1567	2128	2710	3312	3952	4571	5172	5821
Кількість елементів	100	200	300	400	500	600	700	800	900	1000
Кількість замін	48	55	49	50	56	54	51	50	56	43
Сумарна кількість замін	48	103	152	202	258	312	363	413	469	512
кількість проходів без дії	52	97	148	198	242	288	337	387	431	488
Кількість елементів	10	20	30	40	50	60	70	80	90	100
Кількість замін	4	6	4	6	4	9	3	5	7	4
Сумарна кількість замін	4	10	14	20	24	33	36	41	48	52
кількість проходів без дії	6	10	16	20	26	27	34	39	42	48

Таблиця 4.4 — Дослідження стохастичного алгоритму сортування на 100 000 елементах

100000										
Кількість елементів	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Кількість замін	4821	4659	4599	4402	4174	4026	3952	3801	3676	3452
Сумарна кількість замін	4821	9480	14079	18481	22655	26681	30633	34434	38110	41562
кількість проходів без дії	5179	10520	15921	21519	27345	33319	39367	45566	51890	58438
Кількість елементів	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Кількість замін	520	486	542	479	467	538	491	514	486	491
Сумарна кількість замін	520	1006	1548	2027	2494	3032	3523	4037	4523	5014
кількість проходів без дії	480	994	1452	1973	2506	2968	3477	3963	4477	4986
Кількість елементів	100	200	300	400	500	600	700	800	900	1000
Кількість замін	43	50	56	59	48	49	47	43	52	45
Сумарна кількість замін	43	93	149	208	256	305	352	395	447	492
кількість проходів без дії	57	107	151	192	244	295	348	405	453	508
Кількість елементів	10	20	30	40	50	60	70	80	90	100
Кількість замін	6	5	3	6	4	4	4	2	6	5
Сумарна кількість замін	6	11	14	20	24	28	32	34	40	45
кількість проходів без дії	4	9	16	20	26	32	38	46	50	55

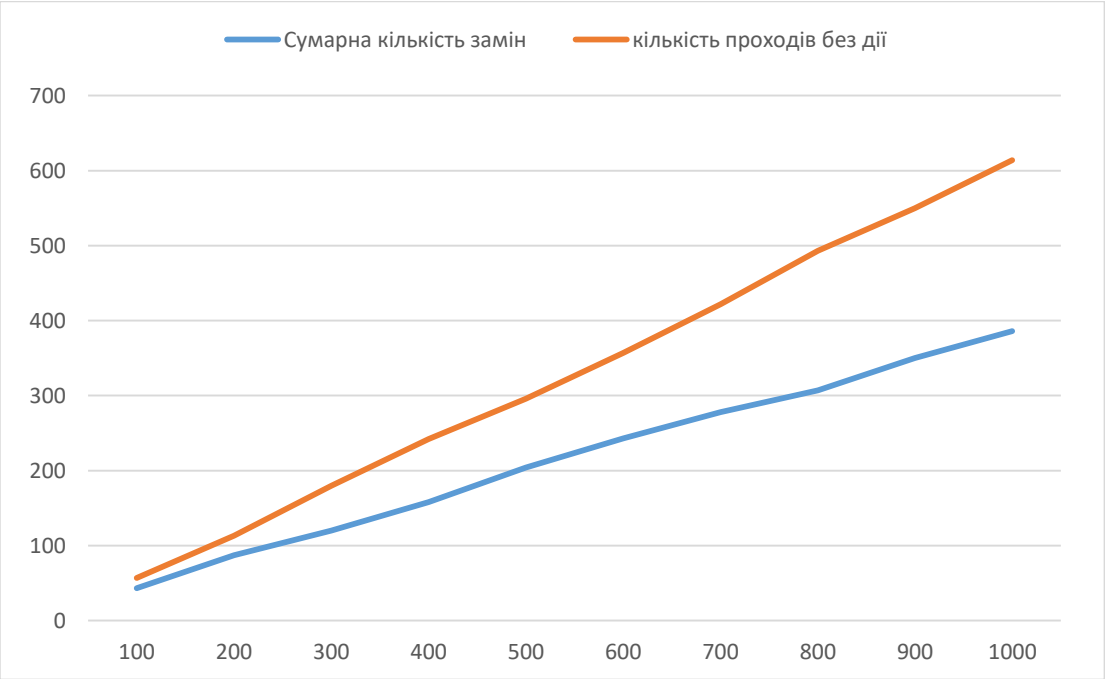

Рисунок 4.2 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 100 на 1000 елементах

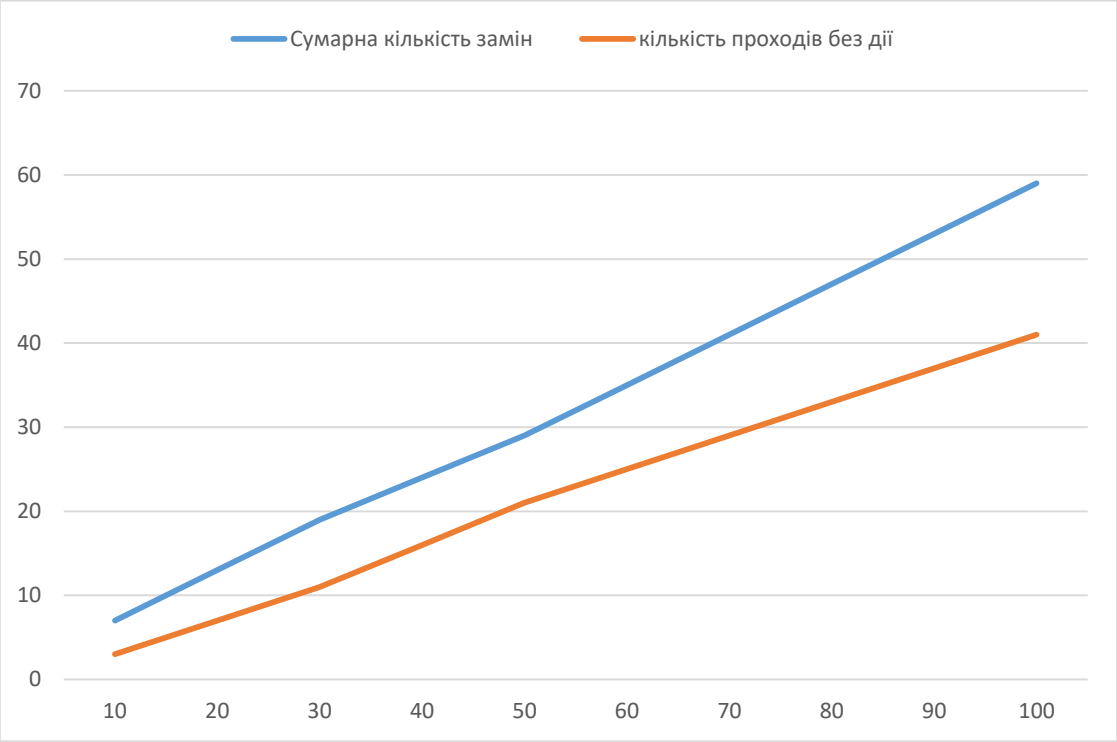

Рисунок 4.3 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 10 на 1000 елементах

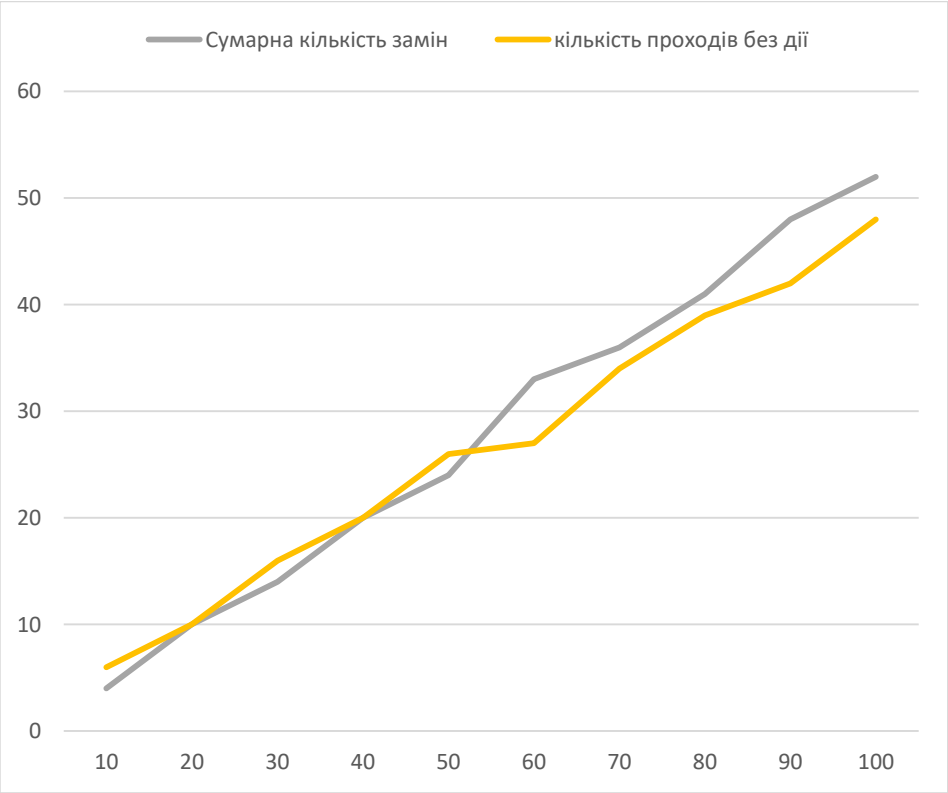

Рисунок 4.4 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 10 на 10000 елементах

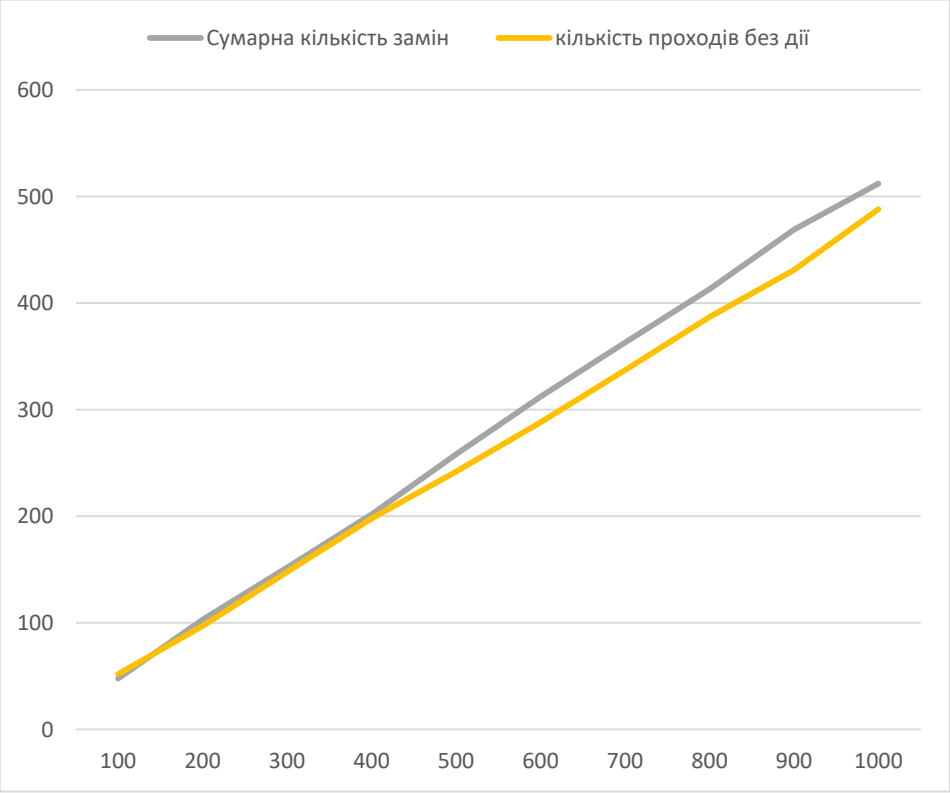

Рисунок 4.5 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 100 на 10000 елементах

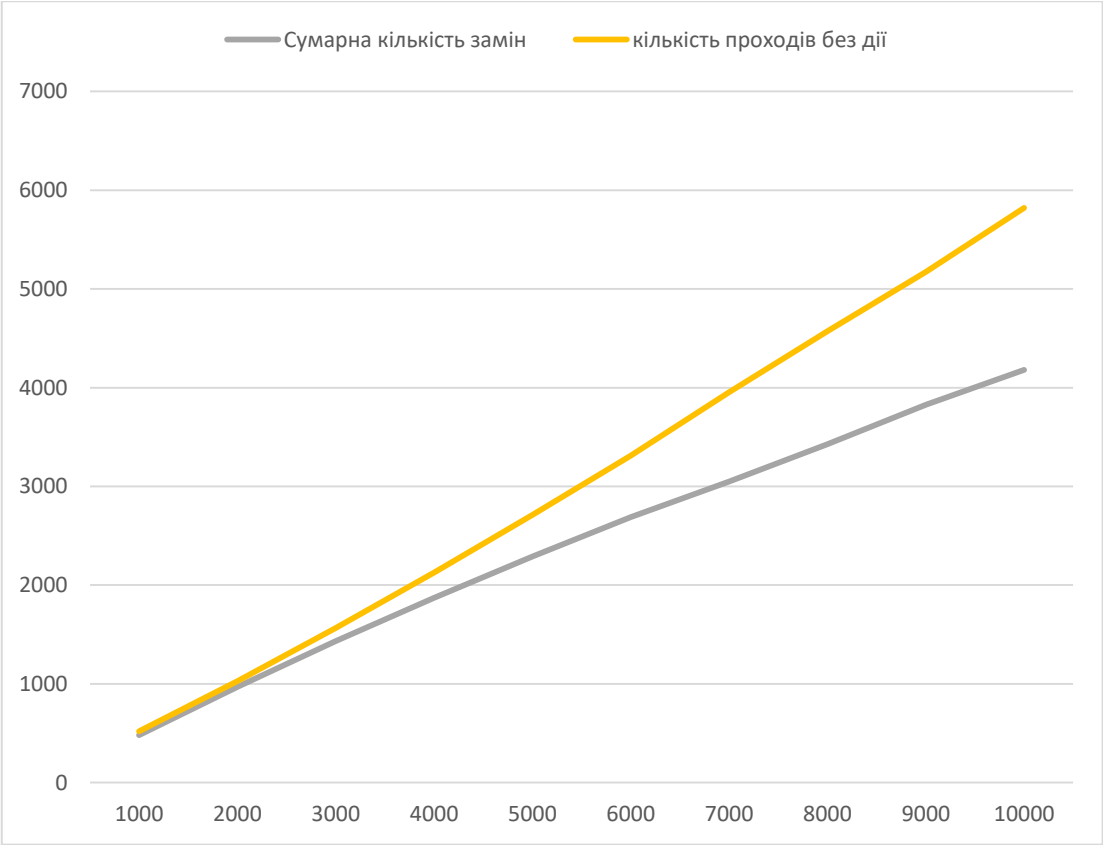

Рисунок 4.19 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 1000 на 10000 елементах

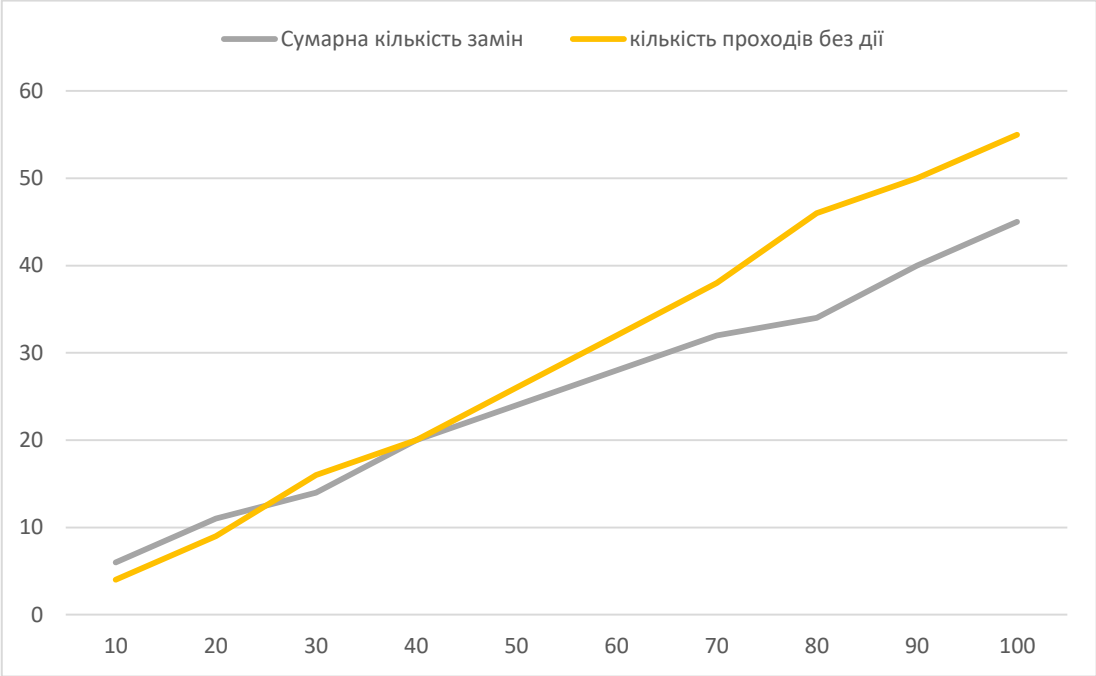

Рисунок 4.6 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 10 на 100000 елементах

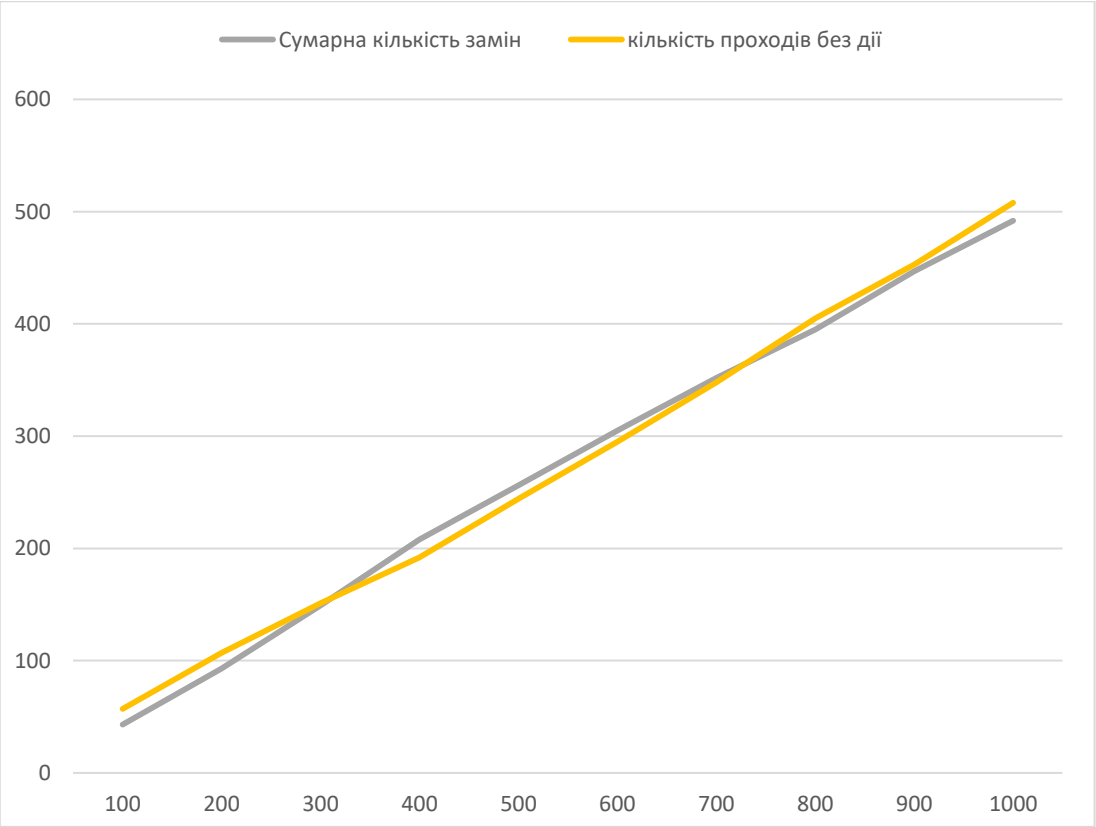

Рисунок 4.7 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 100 на 100000 елементах

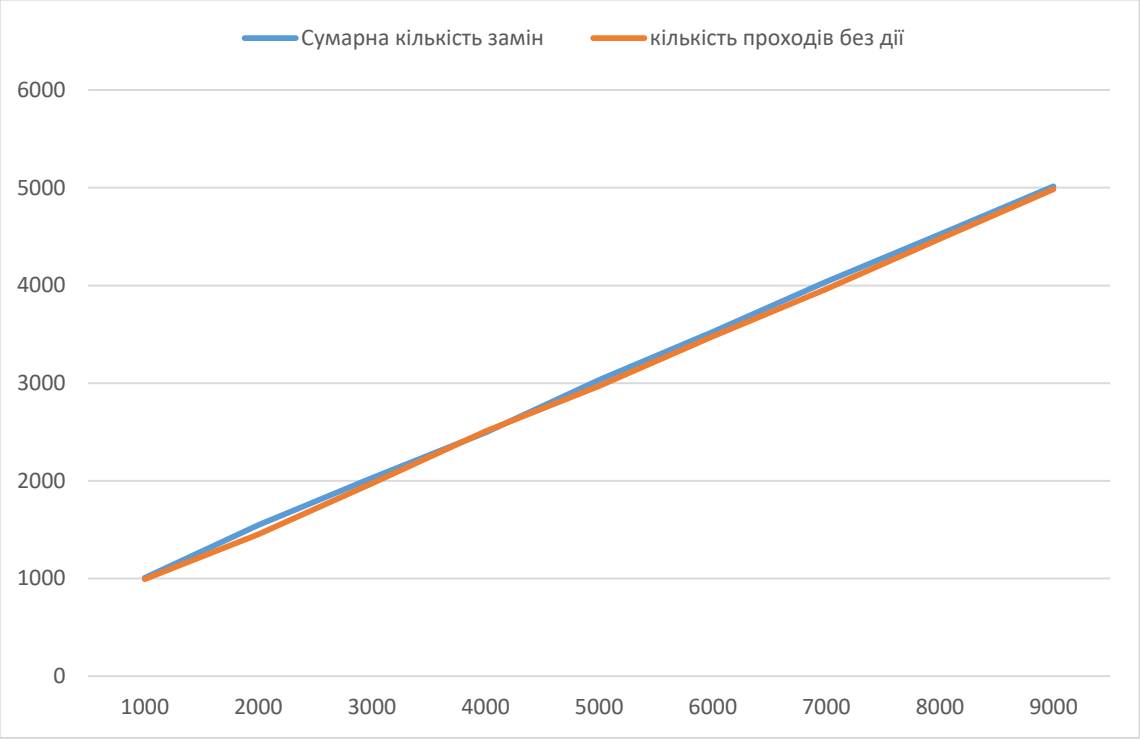

Рисунок 4.8 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 1000 на 100000 елементах

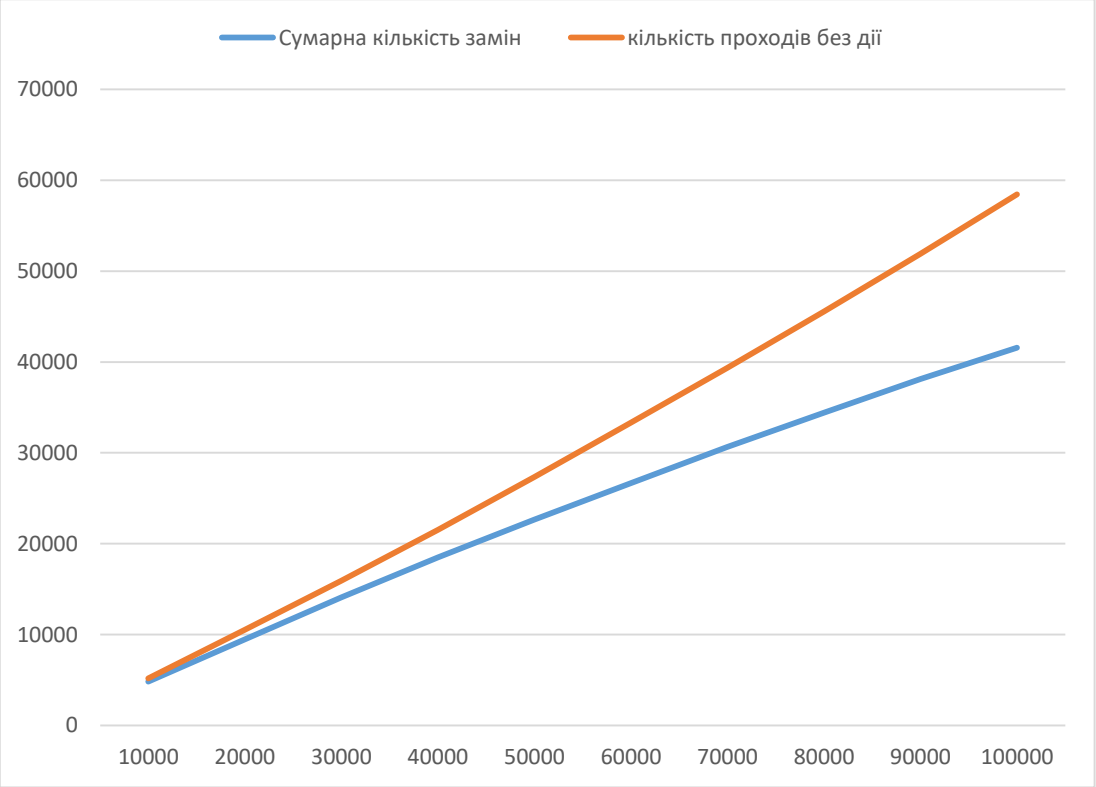

Рисунок 4.9 — Порівняння кількості замін з операціями без заміни з коефіцієнтом 10000 на 100000 елементах

На рис 4.2 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 100 в масиві на 1000 елементів. На ньому чітко видно, що використання жодного коефіцієнта 100 та вище не є доцільним. Кількість корисних операцій, коли виконується заміна буде завжди менше.

На рис 4.3 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 10 в масиві на 1000 елементів. На ньому чітко видно, що використання коефіцієнту від 10 до 100 є доцільним. Очевидне збільшення бачимо на 30 та близько 100. Але враховуючи рисунок 4.2 показник в 100 не й надійним.

На рис 4.4 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 10 в масиві на 10000 елементів. На ньому чітко видно, що використання стає доцільним починаючи з 60.

На рис 4.5 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 100 в масиві на 10000 елементів. На ньому чітко видно, що використання є доцільним, та майже не змінюється до 1000. В другій половині графік показує покращення, але воно досить мале і нівелюється стохастичністю алгоритму.

На рис 4.19 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 1000 в масиві на 10000 елементів. На ньому чітко видно, що використання коефіцієнту значенням 1000 та більше не є доцільним.

На рис 4.6 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 10 в масиві на 100000 елементів. На ньому чітко видно, що використання настільки малого коефіцієнту не є доцільним.

На рис 4.7 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 100 в масиві на 100000 елементів. На ньому лінії йдуть майже паралельно, але покращення бачимо від 300 до 400. Коефіцієнт менше чи більше дасть гірші результати.

На рис 4.8 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 1000 в масиві на 100000 елементів. На ньому чітко видно, що використання коефіцієнту більше 1000 не є ні більш ні менш доцільним. Але дивлячись на рис 4.7 бачимо що й 1000 не самий ефективний коефіцієнт.

На рис 4.9 наведено порівняння кількість замін з кількістю операцій без замін при коефіцієнті кратному 10000 в масиві на 100000 елементів. На ньому чітко видно, що використання коефіцієнту від 10000 не є доцільним. Ефективність алгоритму значно падає.

Далі, що б знайти найкращий коефіцієнт було побудовано графіки, наведені на рис 4.10 – 4.18.

Рисунок 4.10 — Кількість замін на 1000 елементів при коефіцієнті 100

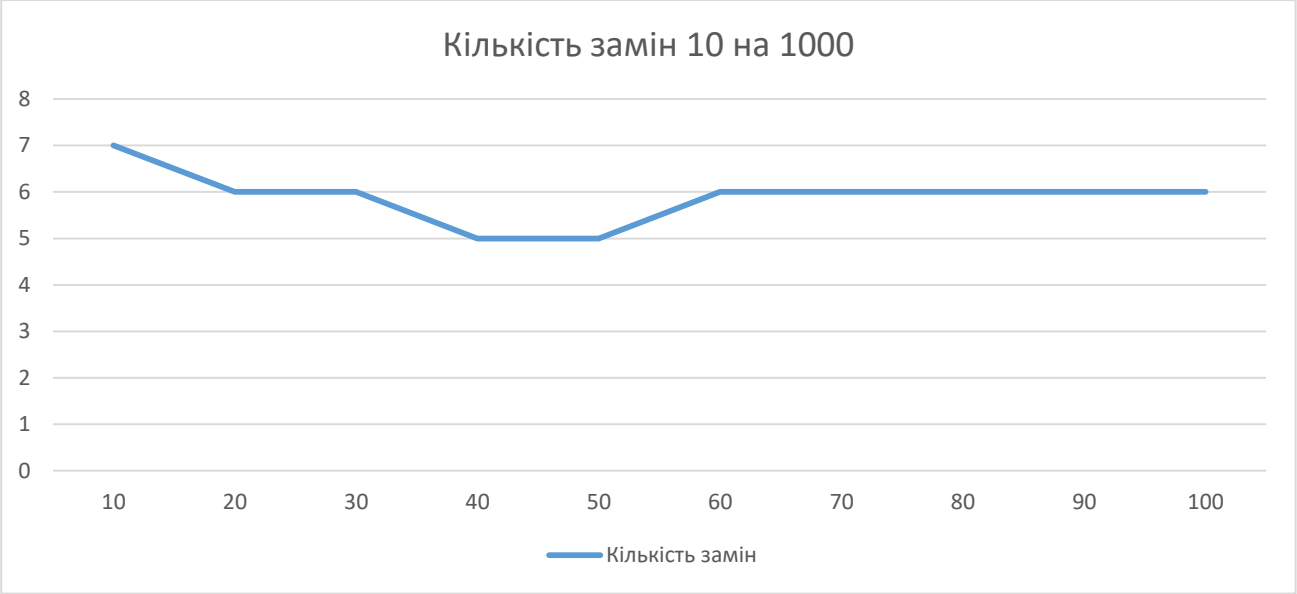

Рисунок 4.11 — Кількість замін на 1000 елементів при коефіцієнті 10

Рисунок 4.12 — Кількість замін на 10000 елементів при коефіцієнті 1000

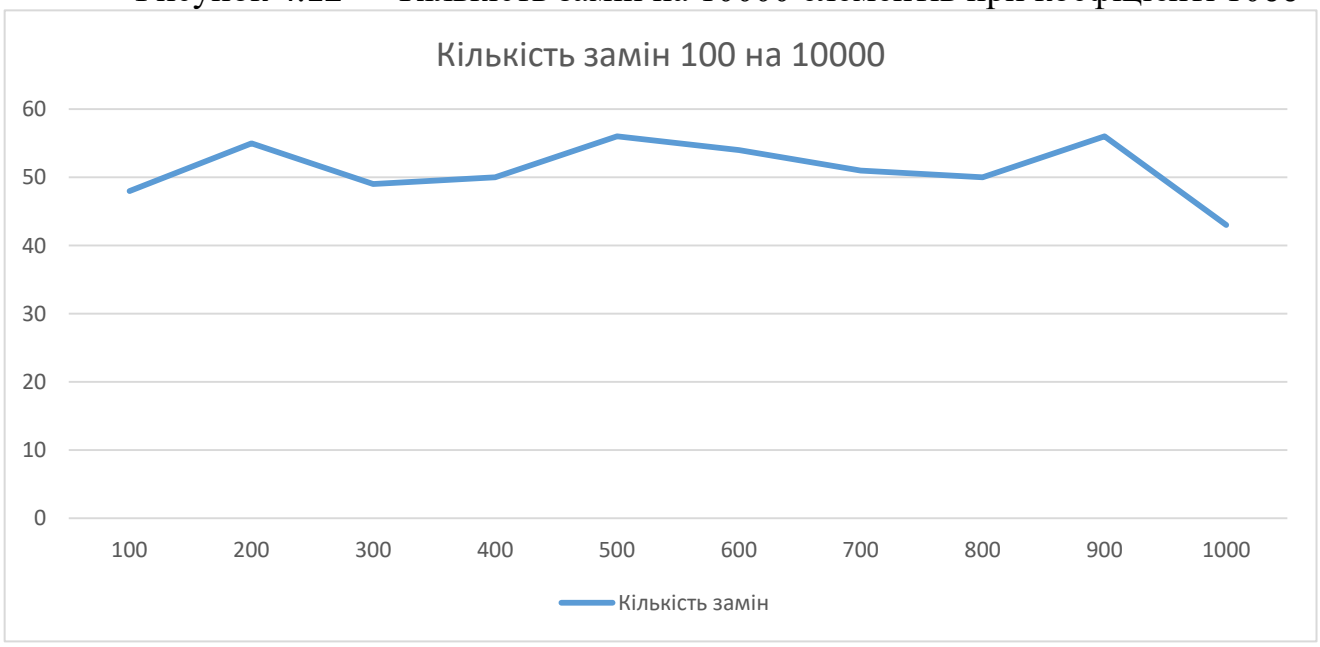

Рисунок 4.13 — Кількість замін на 10000 елементів при коефіцієнті 100

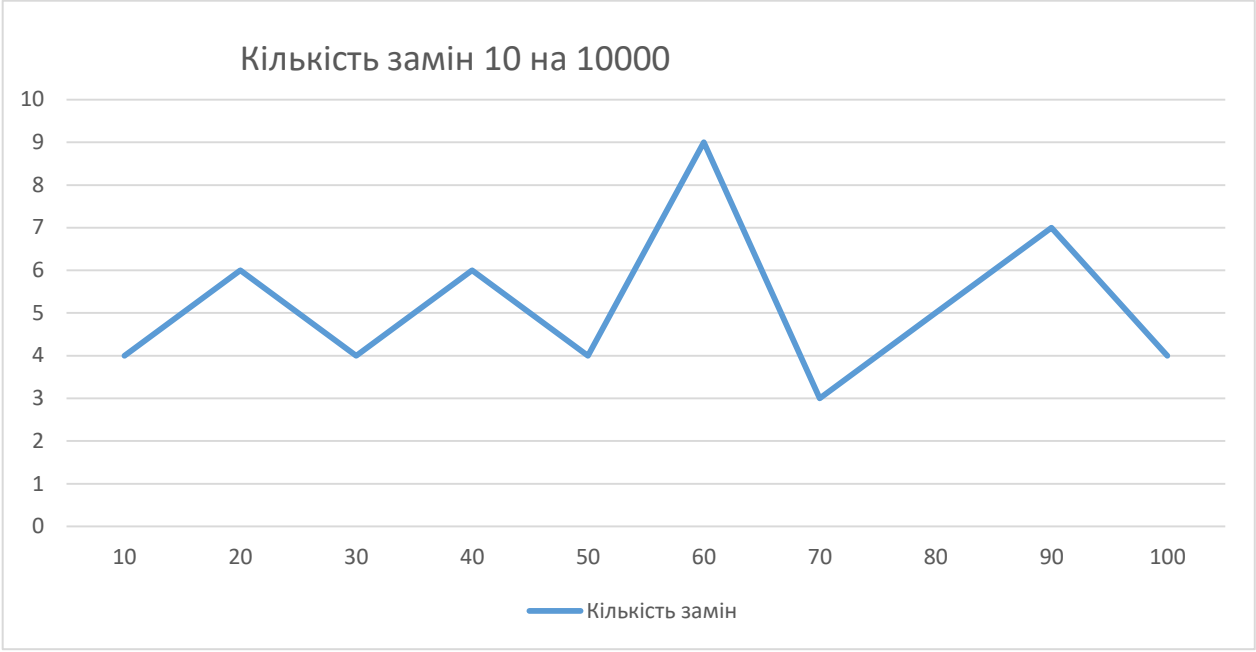

Рисунок 4.14 — Кількість замін на 10000 елементів при коефіцієнті 10

Рисунок 4.15 — Кількість замін на 100000 елементів при коефіцієнті 1000

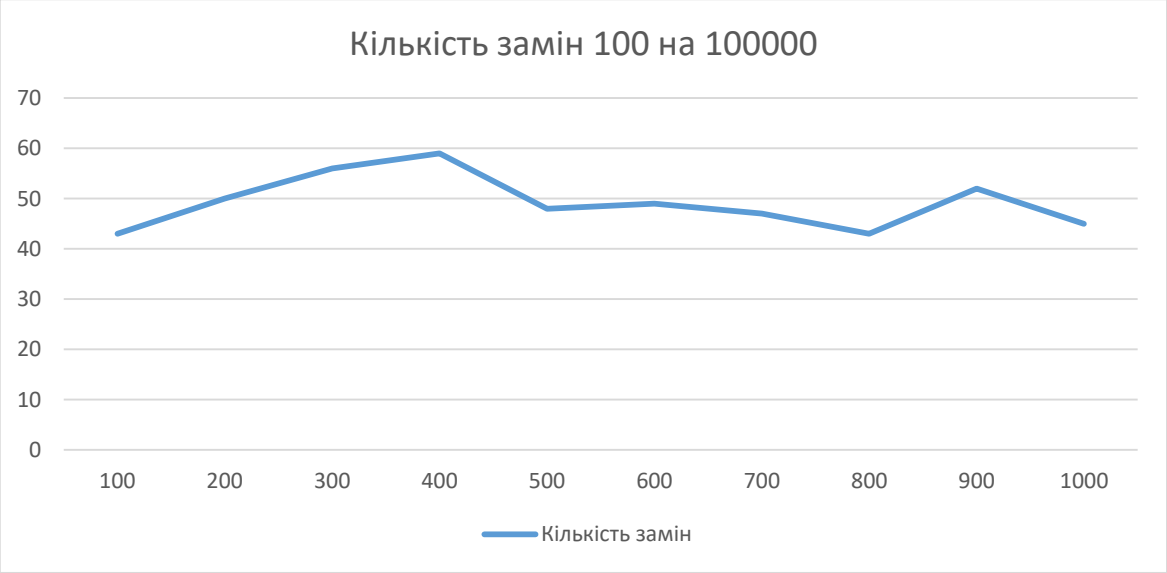

Рисунок 4.56 — Кількість замін на 100000 елементів при коефіцієнті 100

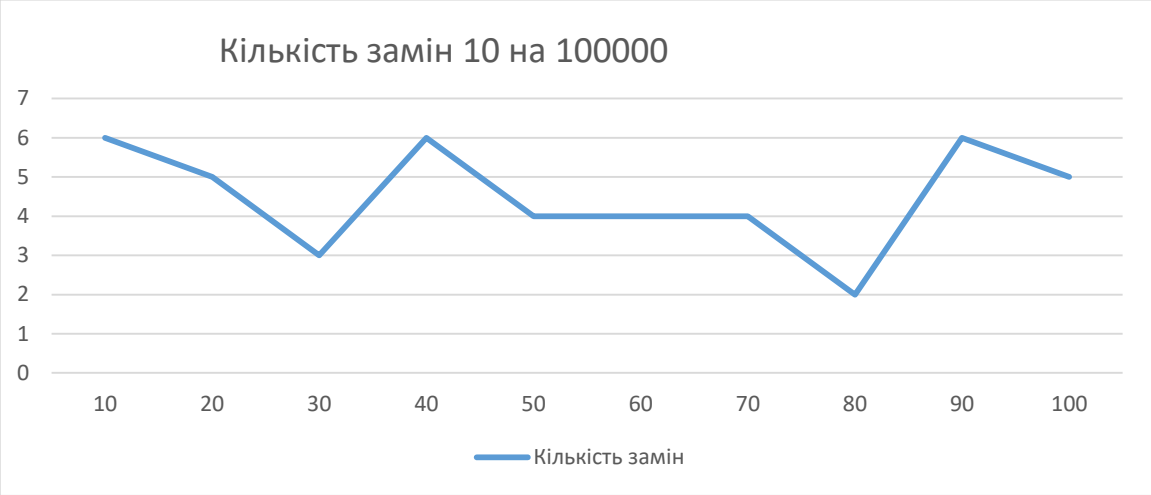

Рисунок 4.17 — Кількість замін на 100000 елементів при коефіцієнті 10

Рисунок 4.18 — Кількість замін на 100000 елементів при коефіцієнті 10000

Для того, що б виконання алгоритму було ефективним, кількість виконаних операцій з заміною має бути більше ніж кількість операцій без заміни. Розглянемо отримані результати.

Рис 4.10 показує, що при коефіцієнті вище 100 в масиві на 1000 елементів корисних операцій буде менше.

Проте на рис 4.11 бачимо, що при 1000 елементах корисним буде використання стохастичного алгоритму з коефіцієнтом до 30. Після, кількість корисних операцій буде або дорівнювати, або нижче.

На рис 4.12 видно, що при розмірі масиву в 10 000 не є доцільним використовувати коефіцієнт вище 1000. Кількість корисних операцій менша за половину.

Рис 4.13 показує, що найвигідніше при 10 000 елементів використовувати коефіцієнт від 100 до 200.

На рис 4.15 бачимо, що при розмірі масиву в 100 000 можна використовувати коефіцієнти до 1000, далі корисність може сильно падати, але багато результатів є корисними. Необхідно розглянути більш широким та вузьким проміжком.

Тому на рис 4.16 розглянемо ефективність на проміжку від 100 до 1000. Тут бачимо, найкориснішим є діапазон від 300 до 400.

А на проміжку від 10 000 до 100 000 бачимо, що ефективність зменшується плавно і починається вона одразу з неефективного варіанту коефіцієнту.

Порівнявши графіки 4.14 та 4.17 бачимо, що коли коефіцієнт дуже низький, алгоритм веде себе дуже нестабільно. Неможливо чітко сказати, що при таких –то значеннях буде відсортовано найбільшу кількість елементів за найменший час.

Отже в процесі порівняння було отримано наступні результати:

- при розмірі масиву в 1 000 елементів корисним є коефіцієнт до 30;
- при розмірі масиву в 10 000 елементів 100-200 найвигідніший коефіцієнт;
- при розмірі масиву в 100 000 елементів, корисний коефіцієнт складає 300-400.

Побудувавши отримані результати на графіку представленому на рисунку 4.20 бачимо, що залежність коефіцієнту приблизно дорівнює кореню від об’єму вхідного масиву. Саме за таку кількість ітерацій розроблений стохастичний алгоритм зробить найбільшу кількість корисних дій та найменше пустих порівнянь.

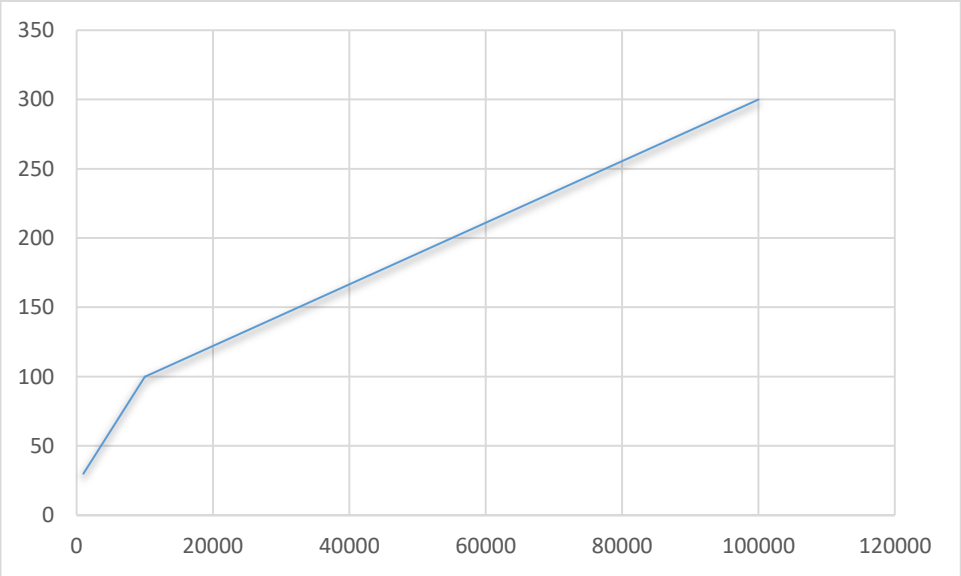

Рисунок 4.20 — Отримані результати для різних об’ємів на графіку.

Проаналізувавши проблему використання алгоритму сортування міхуром було запропоновано реалізувати рекурсивний стохастичний алгоритм сортування. Він полягає у тому, що б стохастичний алгоритм ще раз виконувався, якщо під час минулого проходу виконувались заміни.

Завдяки цьому вдалось покращити швидкодію алгоритму на 100000 елементах майже в 10 разів, що наведено в таблиці 4.5.

Таблиця 4.5 — Результати виконання сортування міхуром та міхуром з стохастичним алгоритмом сортування

Розмір масиву	1000	10000	100000	1000000
Алгоритм сортування	Час виконання, нс			
Міхур	1750776	142569211	16448168694	-
Міхур з стохастикою	1532148	40500235	1606119629	-

Загальні результати виконання всіх обраних алгоритмів наведено в таблиці 4.6. Там демонструється час виконання певних алгоритмів в наносекундах та різниця з стохастико-детермінованим аналогом.

Різниця виражена у відсотках, завдяки чому ми бачимо, що використання стохастичного алгоритму разом з детермінованим є доцільним і ефективним.

В результаті роботи вдалося знайти алгоритм, що виконується навіть швидше за швидке сортування, хоча швидке сортування, на даний момент вважалося найшвидшим у світі.

Таблиця 4.6 — Результати виконання усіх обраних алгоритмів сортування самостійно та з стохастичним алгоритмом на початку.

Розмір масиву	1000	10000	100000	1000000	10000000
Алгоритм сортування	Час виконання, нс				
Міхур	1750776	142569211	16448168694	-	-
Міхур з стохастикою	1532148	40500235	1606119629	-	-
Різниця	12,49%	71,59%	90,24%	-	-
Сортування Тіма	242264	1492901	14262584	124416905	1460718492
Сортування Тіма з стохастикою	236647	1405670	13693951	124014242	1451281480
Різниця	2,32%	5,84%	3,99%	0,32%	0,65%
Плавне сортування	239190	1618754	17344016	218511536	922166782
Плавне сортування з стохастикою	210465	1608730	16876218	204234677	902473192
Різниця	12,01%	0,62%	2,70%	6,53%	2,14%
Швидке сортування	126630	2274900	11313679	104319989	1075398301
Швидке сортування з стохастикою	123866	1717400	10816605	102822558	1043604919
Різниця	2,18%	24,51%	4,39%	1,44%	2,96%
Сортування вставками	282356	47276600	1045130440	-	-
Сортування вставками з стохастикою	237035	42031100	1004145661	-	-
Різниця	16,05%	11,10%	3,92%	-	-
Сортування вибором	496473	43516200	2619710651	-	-
Сортування вибором з стохастикою	527594	41730500	2601862249	-	-
Різниця	-6,27%	4,10%	0,68%	-	-
Сортування злиттям	191080	2774600	14352513	133157727	1563416180
Сортування злиттям з стохастикою	202696	3908300	14112726	133318206	1516474816
Різниця	-6,08%	-40,86%	1,67%	-0,12%	3,00%
Пірамідальне сортування	146738	3032600	14212490	161585603	2838580899
Пірамідальне сортування з стохастикою	133755	2945600	13709181	160566090	2821279824
Різниця	8,85%	2,87%	3,54%	0,63%	0,61%
Сортування Шелла	146302	5217600	16900350	161235879	1941763752
Сортування Шелла з стохастикою	150117	5394200	14948460	154698742	1873640446
Різниця	-2,61%	-3,38%	11,55%	4,05%	3,51%

Висновки до розділу 4

В четвертому розділі було проведено два експерименти з дослідження стохастичних та стохастико-детермінованих алгоритмів сортування. Під час експерименту було зібрано та надано в табличному та графічному виглядах всі необхідні дані.

В результаті проведення експерименту було з’ясовано, що використання стохастичного алгоритму сортування є ефективним лише обмежений проміжок часу.

Приблизний оптимальний коефіцієнт виконання стохастичного алгоритму було виведено, але для покращення швидкодії конкретного алгоритму треба дослідити алгоритм та визначити конкретне значення. Для кожного алгоритму він може значно відрізнятись від виведеного в даній роботі. Наприклад для міхура було виведено рекурсивний алгоритм, з тим самим коефіцієнтом, що покращило швидкодію в 10 разів.

РОЗДІЛ 5.ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Вимоги безпеки під час виконання робіт з розробки програмного забезпечення та виконання досліджень.

Для безпечної та ефективної роботи працівника-програміста необхідно його дотримування до певних правил роботи. Крім цього потрібно, щоб робоче середовище також відповідало визначеним нормам безпеки. В даний момент існує багато нормативно-правових актів з охорони безпеки та державні стандарти, які займаються регулюванням безпеки праці.

Недотримування їх вимог може привести до підвищення ризику впливу шкідливих виробничих факторів на працівника. Тривалий вплив цих шкідливих факторів може негативно вплинути на ефективність праці, або навіть на здоров’я людини.

Оскільки під час розробки деякого програмного забезпечення програміст - розробник так чи інакше працює з комп’ютерною технікою, насамперед потрібно розглянути саме ті нормативні акти та стандарти, які стосуються роботи з відповідною технікою та засобами. Нижче наведені такі документи:

- НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207 (далі скорочено НПАОП 0.700-7.15-18) [41];
- НПАОП 80.0-1.12-04 Правила безпеки під час навчання в кабінетах інформатики навчальних закладів системи загальної середньої освіти, затверджені від 20.11.2007 р. № 252 (далі скорочено НПАОП 80.0-1.12-04) [42];
- ДСанПіН 3.3.2-007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджені від 10.12.1998 р. № 7 (далі скорочено ДСанПіН 3.3.2-007-98) [43];
- Інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури, затверджена від 27.08.2000 р. № 20 [44].

Вимоги з НПАОП 0.700-7.15-18 загалом поділяються на три категорії:

- вимоги безпеки до робочих місць;
- вимоги безпеки під час роботи з екранними пристроями;
- вимоги безпеки до екранних пристроїв.

Вимоги безпеки до робочих місць:

- робочі місця повинні бути спроектовані таким чином, щоб у працівника був простір для рухів та зміни положення;
- випромінювання екранних пристроїв має бути зведено до допустимого рівня;
- освітлення робочого місця повинно створювати необхідний контраст між навколишнім середовищем та екраном та відповідати вимогам ДСанПіН 3.3.2-007-98 [43];
- мікроклімат приміщень з робочими місцями повинен підтримуватися на постійному рівні та відповідати вимогам ДСН 3.3.6.042-99 Санітарних норм мікроклімату виробничих приміщень [45];
- поверхня робочого столу повинна бути з низькою відбивною здатністю, а робочий стіл – достатнього розміру.

Вимоги безпеки під час роботи з екранними пристроями:

- кожний день потрібно прибирати пристрій від пилу або інших забруднень;
- електроні пристрої слід вимикати з електричної мережі в кінці робочого дня;
- також їх слід вимикати у випадку аварійної ситуації;
- не можна виконувати ремонт екранних пристроїв під час роботи на робочому місці:
- не можна працювати з екранними пристроями, які ведуть себе не передбачувано.

Вимоги безпеки до екранних пристроїв:

- екранні пристрої повинні бути безпечними для використання;

- випромінювання екрану повинно бути зведено до незначного рівня;
- усі символи на екрані повинні бути чіткими та помітними;
- екран не повинен миготіти;
- працівник повинен мати змогу змінити яскравість екрану;
- екран не повинен відблискувати;
- клавіатура має бути відокремленою від екрану, щоб робітник мав змогу прийняти зручну позу для роботи;
- поверхня клавіатури теж не повинна відбивати світло, символи на клавіатурі повинні бути чіткими;
- програмне забезпечення екранного пристрою повинно бути простим у використанні для працівника.

Крім того існують й загальні положення про санітарні норми, правила та вимоги, які стосуються робочого процесу не лише програміста-розробника. Ці положення описані в наступних документах:

- Закон України від 14 жовтня 1992 р. № 2694-XII “Про охорону праці”. Редакція від 16.10.2020 [46];
- Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 25.10.2020 [47];
- Закон України від 23 вересня 1999 р. № 1105-XIV “Про загальнообов`язкове державне соціальне страхування”. Редакція від 25.10.2020 [48];
- Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 “Про затвердження Порядку проведення атестації робочих місць за умовами праці”. Редакція від 28.10.2016 [49];
- Постанова Кабінету Міністрів України від 17.04.2019 №337 “Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві” [50];

- НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Зареєстрованого в Мін'юсті України 15.02.2005 за №231/10511. Редакція від 14.04.2017 [51];
- Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дію з 01.03.2019 [52];
- ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень, затверджені Постановою головного санітарного лікаря України №42 від 1 грудня 1999 року [53];
- НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 №1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697 [54];
- Державні санітарні норми виробничої загальної та локальної вібрації ДСН 3.3.6.039-99, початок дії від 01.12.1999 [55];
- Кодекс цивільного захисту України. Редакція від 16.10.2020 [56].

До шкідливих виробничих факторів, які можуть негативно вплинути на програміста-розробника, можна віднести:

- незадовільний мікроклімат (температура, вологість, вентиляція повітря, інфрачервоне або ультрафіолетове випромінювання);
- постійні електричні поля або випромінювання; - небезпечні іонізуючі випромінювання; - високий рівень шумів та вібрацій;
- підвищена запиленість робочого місця;
- ризик отримати удар електричним струмом;
- напруження зорового апарату;
- недостатнє природне або технічне освітлення в робочих приміщеннях.

Мікроклімат приміщення. Мікроклімат – це клімат робочого приміщення, що визначається такими фізичними показниками:

- температура повітря;
- вологість повітря;
- швидкість руху повітря.

Згідно з ДСН 3.3.6.042-99 [53] у табл. 5.1 встановлені оптимальні та допустимі показники мікроклімату у робочому приміщенні.

Таблиця 5.1 – Допустимі показники мікроклімату в приміщенні

Період року	Теплий			Холодний		
Параметри мікроклімату	Температура повітря в приміщенні	Відносна вологість	Швидкість повітря	Температура повітря в приміщенні	Відносна вологість	Швидкість повітря
Оптимальна величина	23-25	40-60%	0,1-0,2 м/с	22-24	40-60%	До 0,1 м/с

Шум погіршує умови праці надаючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах і т.д. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму (вище 80 дБ (А)) на слух людини приводить до його часткової або повної втрати.

У табл. 5.2 вказані граничні рівні шуму, що є безпечними відносно збереження здоров’я.

Таблиця 5.2 – Граничні рівні шуму в дБ на робочому місці

Категорія напруженості праці	Категорія важкості праці			
	Легка	Середня	Важка	Дуже важка
Мало напружений	80	80	75	75
Помірно напружений	70	70	65	65
Напружений	60	60		
Дуже напружений	50	50		

Рівень шуму на робочому місці програмістів не повинен перевищувати 50дБА, а в залах обробки інформації на обчислювальних машинах - 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані 95 звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи. Але в сучасному світі є багато рішень які 97 боряться з шкодою від сидячої роботи. Наприклад стіл який має регулювання по висоті, колінні крісла та інші.

Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Максимальна зона досяжності рук - це частина моторного поля робочого місця, обмеженого дугами, описуваними максимально витягнутими руками при русі їх у плечовому суглобі.

Оптимальна зона - частина моторного поля робочого місця, обмеженого дугами, описуваними передпліччями при русі в ліктьових суглобах з опорою в точці ліктя і з відносно нерухомим плечем.

Також одним з найважливіших аспктів захисту праці є освітлення.

При недостатньому освітленні зорове сприйняття людини знижується, що спричиняє головний біль, та хвороби очей. Через постійне напруження очі перевтомлюються, що негативно впливає на їх стан.

Джерела світла поділяють на наступні:

- природні, світло що потрапляє до нас від сонця, небосхила;
- штучні, світла, яке генерується електричними пристроями.

Штучні також поділяються на місцеві, загальні, та комбіновані.

Використання тільки місцевого освітлення заборонено через небезпеку травматизму та професійних захварювань. Людина, через погане освітлення, може нахилятися до обладнання, що може призвести до травм.

Працівник програміст працює з літерами, що, зазвичай 3-5 міліметрів. Така робота відноситься до третього разряду зорової роботи. Згідно з ДБН В.2.5-28:2018 [52] загальна освітленість має тавити 200 лк, комбінована – 400 лк.

За допомого додатку люксометру для мобільного телефону було визначено, що загальна освітленість становить 226 лк, місцева освітленість – 330 лк. Місцева освітленість виявилася недостатньою, через що було замінено лампочку. Новий показник становить 387 лк. Отже освітлення задовольняє вимогам.

5.2 Дії працівників в надзвичайних ситуаціях

Наразі для робітника-програміста найбільш актуальною з надзвичайних ситуацій є пожежа. Правила пожежної безпеки затверджені наказом МВС від 30.12.2014[55], де приведено порядок дій у разі виникнення пожежі.

При виявленні перших ознаки задимлення або пожежі, необхідно:

- зателефонувати за номером «101». Потрібно назвати свої данні та надати про місце пожежі, кількість поверхів та іншу інформацію;
- у разі якщо пожежа сталася на підприємстві, слід негайно повідомити про неї черговому чи директору.;
- допомогти людям покинути приміщення та надати першу допомогу особам які постраждали;
- викликати іншу служби порятунку якщо в цьому є потреба.

Якщо пожежа виникла на підприємстві, службова особа об’єкта, яка прибула до місця пожежі, має:

- викликати рятувальну службу цивільного захисту якщо цього ще не зробили інші;
- скласти план дій оцінивши ситуацію и ступінь загрози;
- вимкнути електроживлення щоб запобігти розповсюдженню пожежі, та інші прилади. Припинити всі роботи які не відносяться до усунення пожежі;
- увімкнути пожежну тривогу;
- провести евакуацію людей;
- допомагати рятівникам при потребі. Допомагати рятувальникам при виборі найкоротшого шляху при ліквідації, представити план будівлі та інше.

Також критичною є ситуація ураження електричним струмом. Правила безпечної експлуатації наведено в НПАОП 40.1–1.21–98[57]. Згідно з ними на підприємстві мають бути відповідальні люди та правила з охорони праці які включають в себе поведінку при ураженні персоналу електричним струмом.

Основними причинами електротравматизму на виробництві є:

- випадкове доторкання до неізольованих струмопровідних частин електроустаткування;
- використання несправних ручних електроінструментів;
- застосування нестандартних або несправних переносних світильників напругою 220 чи 127 В;
- робота без надійних захисних засобів та запобіжних пристосувань;
- доторкання до незаземлених корпусів електроустановок, що опинилися під напругою внаслідок пошкодження чи пробою ізоляції;
- недотримання правил будови, улаштування, безпечної експлуатації електроустановок та правил експлуатації електрозахисних засобів тощо.

Основними діями в такій ситуації є[58]:

- якщо можливо, припинити постачання струму на потерпілого не приближаючись до джерела витоку(вимкнувши електроенергію в мережі);
- викликати лікаря;
- надати долікарську допомогу.

Заходи залежать від стану в якому перебуває потерпілий.

Якщо потерпілий при свідомості його треба покласти на підстіоку з тканини або одяг, розстібнути одяг, що перешкоджає вільному диханню та забезпечити спокій до приїзду лікаря.

Якщо потерпілий непритомний, йому треба дати вдихнути нашатирний спирт або збризнути обличчя холодною водою. Після того як потерпілий прийшов до тями необхідно дати випити 15-20 крапель валеріани та гарячий чай.

Якщо потерпілий не має ознаків життя, необхідно одразу почети серцево-легеневу реанімацію. Перед СЛР необхідно виконати наступне:

- покласти спиною на тверду рівну поверхню;
- звільнити від одягу, що стискає;
- підкласти під лопатки валик з будь якого матеріалу;
- відхилити голову назад.

СЛР складається з непрямого масажу серця та штучного дихання.

Під час непрямого масажу серця рятівник стає збоку від потерпілого, поклавши кисті рук на нижню третину грудної клітки та енергійними поштовхами натискує на неї. Необхідна частота становить 60 – 65 натискань у хвилину.

Заходи необхідно виконувати одночасно. Якщо їх виконує одна людина, необхідно додержуватись послідовності два глибоких видохи, 15 натискань, і повторити.

Заходи вважаються вдалими, якщо звузились зіниці, почала рожевіти шкіра, у першу чергу верхньої губи, відчувається пульс на сонній артерії. Продовжувати необхідно доки у потерпілого не відновиться дихання та робота серця, або до приїзду швидкої допомоги.

Висновки до розділу 5

В даному розділі було розглянуто основні положення з захисту здоров’я та життя під час розробки програмного забезпечення.

Було розглянуто основні норми та документи, які визначають вимоги до робочого процесу. Було визначено основні показники мікроклімату та шуму на робочому місці безпечні для виконання роботи. Було розглянуто основні вимоги до екранних пристроїв.

Також було розглянуто основні правила поведінки в надзвичайних ситуаціях. Було визначено, що зараз актуальною надзвичайною ситуацією є пожежа. Було проаналізовані актуальні нормативні акти, з яких виведено основні дії при виникненні подібних ситуацій.

ВИСНОВКИ

Результатом виконання даної дипломної роботи є визначення ефективності використання стохастики як самостійно, так і з детермінованими алгоритмами сортування. Було досліджено виконання стохастичних та стохастико-детермінованих алгоритмів сортування на перевірку покращення швидкодії.

Після детального аналізу предметної сфери було визначено, що стохастика ніяк не використовується в алгоритмах сортування, хоча в інших сферах вдало використовується значно покращуючи необхідні результати виконання алгоритмів. Саме тому було обрано використати стохастику, для покращення часової ефективності сортування.

Для подальшого дослідження було обрано наступні алгоритми сортування:

- стохастичний(спеціально розроблений);
- сортування міхуром;
- швидке сортування;
- сортування вставкою;
- сортування злиттям;
- сортування вибором
- сортування Тіма;
- плавне сортування;
- сортування Шелла.

Для дослідження часових характеристик виконання алгоритмів було розроблено спеціальний додаток. Кожен з алгоритмів було реалізовано у додатку мовою Java. Додаток надає можливість користувачу обирати алгоритм, кількість експериментів, розмір масиву, найбільше та найменше значення.

Було поставлено два експерименти. Один, щоб з’ясувати чи є доцільним використання стохастичного алгоритму у чистому вигляді. Другий, щоб з’ясувати, чи можна використовуючи стохастичний алгоритм покращити часові характеристики детермінованих алгоритмів.

Отримані дані експерименту було отримано та проаналізовано. На основі отриманих даних першого експерименту проводився другий експеримент. А саме в першому експерименті було визначено середній коефіцієнт, при якому найбільша швидкодія стохастичного алгоритму, при найбільшій кількості замін.

Отримані дані кажуть про наступне:

- стохастичний алгоритм не доцільно використовувати для строгого сортування;
- стохастичний алгоритм ефективно використовувати обмежено, за визначеною в даній роботі кількістю разів, а саме корінь з розміру вхідного масиву;
- стохастичний алгоритм добре показує себе при первинній обробці масиву, що значно покращує результати разом з детермінованим алгоритмом сортування.

В даній роботі було виведено декілька найефективніших алгоритмів сортування, і алгоритми з використанням стохастики виявилися найшвидшими.

Крім того, було розглянуто основні аспекти захисту життя і здоров’я під час роботи згідно з нормативними актами, та поведінку працівника під час надзвичайних ситуацій.

РЕКОМЕНДАЦІЇ

В результаті роботи було покращено швидкодію існуючих алгоритмів. Найшвидшими стали швидке сортування та плавне сортування з використанням стохастики. Також розроблений додаток дуже зручний та ефективний при дослідженні.

Завдяки розробленому додатку можна і надалі виконувати дослідження, додавати алгоритми, способи їх взаємодії. Також, імпортувавши в свій проект клас Experiment.class можна використовувати розроблені алгоритми у своїх проектах.

Але робота з стохастикою не завершена. Надалі треба обрати один чи декілька алгоритмів, і модифікувати стохастичний алгоритм індивідуально під нього. Саме тоді можна буде отримати значно кращі результати, як це було зроблено з сортуванням міхуром.

СПИСОК ДЖЕРЕЛ

1. Стаття «Anonymity in a big data world» [Електронне джерело] – 2021 URL-— https://www.klcommunications.com/anonymity-in-a-big-data-world/
2. Стаття «Знай складність алгоритмів» [Електронне джерело] – 2021 URL — https://habr.com/ru/post/188010/
3. Стаття «Стохастичність» [Електронне джерело] URL — https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D1%85%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C
4. Стаття «Обираємо найкращій алгоритм» [Електронне джерело] – 2021 URL — https://habr.com/ru/post/133996/
5. SOLID принципи [Електронне джерело] URL — https://ru.wikipedia.org/wiki/SOLID_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)
6. Шаблони проектування [Електронне джерело] URL — https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#%D0%A2%D0%B8%D0%BF%D1%8B_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F

7. Функціональне тестування [Електронне джерело] URL — https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5
8. Інтеграційні тести [Електронне джерело] URL — https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5
9. Проведення експерименту [Електронне джерело] URL — https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%8D%D0%BA%D1%81%D0%BF%D0%B5%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D1%82%D0%B0#%D0%AD%D1%82%D0%B0%D0%BF%D1%8B_%D0%BF%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D1%8D%D0%BA%D1%81%D0%BF%D0%B5%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D1%82%D0%B0
10. Лінійний стохастичний алгоритм сортування [Електронне джерело] – 2021 URL — https://ela.kpi.ua/bitstream/123456789/10591/1/15_p63.pdf
11. Ященко В. В. Введення в криптографію В. В. Ященко. М.: МЦНМО, 2000.
12. Пугачев В. С. Теорія випадкових функцій та її використання. - М. Физматгиз, 1962. – 720 с
13. Одинцов Б.Е. Зворотні обчислення в формуванні рішень економіки – Финансы и статистика, 2004. – 256 с
14. Виштак О.В. Использование технологии обратных вычислений при мониторинге качества дополнительного образования в вузе / И.А. Штырова // Вестник Астрахан. гос. техн. ун-та. – 2014. – № 2. – С. 67–73.

15. Бармина Е.А. Мониторинг качества коммерческой организации. Структурирование показателей. Применение когнитивных карт / И.Ю. Квятковская //Вестник Астрахан. гос. техн. ун-та. – 2010. – № 2. – С. 15–20.
16. Мартьянова А.В. Управление эффективностью банка на базе обратных вычислений // Вестник магистратуры. – 2015. – № 6(45). – С. 77–79.
17. Стохастичні алгоритми вирішення зворотних задач [Електронне джерело] – 2021 URL —https://journal.tusur.ru/storage/57442/23-%D0%93%D1%80%D0%B8%D0%B1%D0%B0%D0%BD%D0%BE%D0%B2%D0%B0.pdf?1489569522
18. Грибанова Е.Б. Решение обратных задач экономики с помощью модифицированного метода обратных вычислений // Проблемы управления. – 2016. – №5. – С. 35–40
19. M. Kac & J. Logan, in Fluctuation Phenomena, eds. E.W. Montroll & J.L. Lebowitz, North-Holland, Amsterdam, – 1976
20. E. Nelson, Quantum Fluctuations, Princeton University Press, Princeton, – 1985
21. Jeff Miller et al. Earliest Known Uses of Some of the Words of Mathematics (S).
22. Douglas Hubbard «How to Measure Anything: Finding the Value of Intangibles in Business» pg. 46, John Wiley & Sons, – 2007
23. Priplata A. et al. Noise-Enhanced Balance Control in Patients with Diabetes and Patients with Stroke. Архивная копия от 23 сентября 2015 на Wayback Machine Ann Neurol.
24. Красовский Г.И., Филаретов Г.Ф. Планирование эксперимента. — Минск: Изд-во БГУ, 1982. — 302 с.
25. Ермаков С. М. Математическая теория планирования эксперимента. — М.: Наука, 1983. — 392 с.
26. Григорьев Ю. Д. Методы оптимального планирования эксперимента: линейные модели: Учебное пособие. — СПб.: Издательство «Лань», 2015. — 320 с.

27. Павлюк О.В. Ефективний синтаксичний аналіз та розпізнавання структурованих зображень / О.В. Павлюк, Б.Д. Савчинський // Управляющие системы и машины. – 2005. – № 5 . – С. 13 - 24.
28. Костенко В.А. Оценка сложности и качества различных итерационных алгоритмов построения расписаний / В.А. Костенко // Искусственный интеллект. – 2004. – №2. – С. 101-104.
29. Arora S. Computational complexity: a modern approach / S. Arora, B. Barak. – N.Y.: Cambridge University Press, 2009. – 594 p.
30. Du Ding-Zhu Teory of computational complexity / Ding-Zhu Du, Ker-I. Ko. – N.Y.: Wiley Cop., 2000. – 491 p.
31. Goldreich O. Computational Complexity / A conceptual perspective / O. Goldreich. – N.Y.: Cambridge University Press, 2008. – 606 p.
32. Hartmanis J. Feasible computations and provable complexity properties / J. Har-tmanis. – Bristol: Arrowsmith Ltd., 1989. – 62 p.
33. Melkebeek D. Randomness and Completeness in computational complexity / D. Melkebeek. – N.Y.: Springer-Verlag, 1999. – 196p.
34. Sipser M. Introduction to the theory of computation / M. Sipser. – [Second edition.] – Boston: Thomson, – 431 p.
35. Дорошенко А.Е. Использование комплексных функциональных устройств микропроцессоров при оптимизирующей компиляции / А.Е. Дорошенко, Д.В. Рагозин // Управляющие системы и машины. – 2004. – № 2 . – С. 46-55.
36. Кнут Д. Искусство программирования, том 1. Основные алгоритмы / Д. Кнут. – [3-е изд.]. – М.: Издательский дом "Вильямс", 2000. – 720 с.
37. Кнут Д. Э. Искусство программирования, том 3. Сортировка и поиск / Д. Кнут. – [3-е изд.]. – М.: Издательский дом "Вильямс", 2000. – 832 с.
38. Котов В.Е. Теория схем программ / В.Е. Котов, В.К. Сабельфельд. – М.:Наука, 1991. – 248 с.

39. «The Digitalization of the World» [Електронне джерело] – 2021 URL — https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf – 6 с.
40. Збірник тез доповідей 81 всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту» 28 жовтня 2021 року – 4 с.
41. НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207.
42. Інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури, затверджена від 27.08.2000 р. № 20.
43. ДСН 3.3.6.042-99, Державні санітарні норми мікроклімату виробничих приміщень затверджені від 01.12.99 р. № 42.
44. Закон України "Про охорону праці" прийнятий від 14 жовтня 1992 року № 2695-12.
45. Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 25.10.2020.
46. Закон України від 23 вересня 1999 р. № 1105-XIV "Про загальнообов`язкове державне соціальне страхування". Редакція від 25.10.2020.
47. Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 "Про затвердження Порядку проведення атестації робочих місць за умовами праці". Редакція від 28.10.2016.
48. Постанова Кабінету Міністрів України від 17.04.2019 №337 "Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві".
49. НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Зареєстрованого в Мін'юсті України 15.02.2005 за №231/10511. Редакція від 14.04.2017.

50. Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дію з 01.03.2019.
51. ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень, затверджені Постановою головного санітарного лікаря України №42 від 1 грудня 1999 року.
52. НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 №1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697.
53. Державні санітарні норми виробничої загальної та локальної вібрації ДСН 3.3.6.039-99, початок дії від 01.12.1999.
54. Кодекс цивільного захисту України від 02 жовтня 2012 року № 5403-VI. Редакція від 16.10.2020
55. ДСТУ ISO 6309:2007 «Протипожежний захист. Знаки безпеки»
56. Кодекс цивільного захисту України від 02 жовтня 2012 року № 5403-VI. Редакція від 16.10.2020
57. НПАОП 40.1–1.21–98 Правила безпечної експлуатації електроустановок в Україні, затверджений наказом Державного комітету України по нагляду за охороною праці від 16.03.94 і зареєстрованого в Міністерстві юстиції України 12.05.94
58. Електробезпека [Електронне джерело] URL — http://www.ztec.com.ua/ztec/e-lib/%D0%9E%D1%85%D0%BE%D1%80%D0%BE%D0%BD%D0%B0%20%D0%BF%D1%80%D0%B0%D1%86%D1%96/%D0%A2%D0%B5%D0%BC%D0%B0%2017%20%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0.pdf

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис Боднар

ПЛАТФОРМА ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО-ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01198-01-ЛЗ

Завідувач кафедри КІТ
доц. Вадим Горячкін

Керівник розробки
проф. Віктор Шинкаренко

Виконавець
студент групи ПЗ2021
Костянтин Галаган

Нормоконтролер
доц. Олена Куроп'ятник

2021

ЗАТВЕРДЖЕНО

01116130.01198-01

ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

01116130.01198-01

Аркушів 19

2021

АНОТАЦІЯ

Документ 1116130.01198-01 «Платформа дослідження стохастичних та стохастико-детермінованих алгоритмів сортування. Технічне завдання» входить до складу програмної документації до дипломного проекту.

У даному документі представлене призначення та область застосування програмного продукту, основні вимоги, стадії та строки виконання проекту, техніко-економічні показники, що пред’являються до програмного продукту.

ЗМІСТ

ВСТУП

Платформа для дослідження стохастичних та стохастико-детермінованих алгоритмів сортування — програмний продукт для користування через будь яке середовище розробки з підтримкою мови Java.

Для управління додатком буде розроблено окремий клас, через який буде виконуватись взаємодія з алгоритмами. Програмним продуктом одночасно буде користуватись одна людина. Результати роботи програмного продукту не будуть впливати на наступне його використання..

Додаток може бути масштабовано для збільшення області застосування в будь-яких університетах та інших закладах освіти додаванням нового класу, який містить інші алгоритми, способи взаємодії, або важливі дані, та використання його в головному класі як об’єкт.

1. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки даної платформи є наказ ректора Українського державного університету науки і технологій.

Тема розробки: Автоматизована платформа дослідження алгоритмів сортування.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення.

Автоматизована платформа дослідження стохастичних та стохастико-детермінованих алгоритмів реалізує повний цикл заходів необхідний для точного дослідження часових характеристик алгоритмів. В програмному продукті наявні алгоритми сортування, але можливе дослідження і виконання інших алгоритмів не пов’язаних з сортуванням

Експлуатаційне призначення.

Створення системи викликано необхідністю мати завершений інструмент з реалізованими алгоритмами сортування та різними способами їх взаємодії, що пришвидшить подальше дослідження стохастичної складової у покращенні швидкодії існуючих алгоритмів сортування.

3. ВИМОГИ ДО ПРОГРАМИ

3.1. Вимоги до функціональних характеристик

Продукт повинен являти собою систему з підтримкою обробки великих масивів даних обраним алгоритмом. Серед необхідних функцій:

- наявність основних існуючих алгоритмів сортування та стохастичного, для дослідження їх виконання;
- можливість виконати сортування заданим алгоритмом та отримати в результаті відсортований масив даних.

Формат вхідних даних – розмір масиву даних, найбільше та найменше значення елементів масиву, вхідний масив для сортування, кількість повторювань експерименту та алгоритм сортування.

Формат вихідних – середня кількість часу виконання сортування заданим алгоритмом, кількість операцій заміни виконана стохастичним алгоритмом та відсортований масив даних.

3.2. Вимоги до надійності

Надійність системи визначатиме її безпосередня точність виконання сортування та дослідження часових характеристик алгоритмів, адже недостовірність вихідних даних може призвести до фінансових збитків через впровадження неоптимальних алгоритмів сортування або некоректність подальших досліджень зроблених через розроблюваний додаток.

Наявність архівної копії тексту програми на зовнішньому носії.

Умови експлуатації

Експлуатація передбачає роботу на персональному комп’ютері у середовищі розробки з підтримкою мови програмування Java, на будь якій операційній системі.

Апаратна частина представлена комп’ютерами, що розташовані в приміщенні. Температура експлуатації від 20 С до 30 С, відносна вологість 40-60%.

3.3. Вимоги до складу і параметрів технічних засобів

Програмний продукт розрахований для функціонування на ПК, що має такі характеристики:

- маніпулюючий пристрій миша;
- клавіатура.

3.4. Вимоги до інформаційної і програмної сумісності

Користувачу для використання додатку необхідний будь-яке сучасне середовище розробки з підтримкою мови програмування Java 5 версії.

3.5. Вимоги до маркування і упаковки

Передача офлайн версії продукту також потребує унікального маркування: «Платформа дослідження стохастичних та стохастико-детермінованих алгоритмів сортування»

Упакування офлайн версії проходитиме після проходження контролю якості командою тестувальників, а також пробного запуску на змодельованому сервері безпосередньо в компанії.

3.6. Вимоги до транспортування і зберігання

Продукт буде розгорнуто безпосередньо на комп’ютері компанії, тому транспортування проходитиме на носії у вигляді CD диску.

Зберігання продукту здійснюватиметься на фізичних носіях у вигляді CD дисків, крім цього локальну копію розробленого продукту буде передано на жорстокому USB-носії при передачі продукту за контрактом.

4. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- технічне завдання;
- робочий проект.

Робочий проект складається з:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача. Керівництво з проведення експериментів.

5. КОШТОРИС НА РОЗРОБКУ ПЗ

Основна мета розробки техніко-економічного обґрунтування (ТЕО) – дати фінансову оцінку передбачуваних витрат та одержуваного корисного результату, а також оцінити прибутковість проекту і, в кінцевому підсумку, економічну доцільність його розробки та впровадження.

Початковим етапом розрахунку величини трудових витрат розробників є оцінка розміру програмного забезпечення. Основні відмінності методик, що застосовуються в оцінці трудовитрат, полягають у використовуваному типі критерію оцінки якості (кількісний або якісний) [7].

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях.

$$E = a \cdot S^b \cdot EAF, \tag{1}$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, a = 2,4; b = 1,05

Припустимо, що розмір програмного коду програмного засобу – 630 рядків:

$$E = 2{,}4 \cdot 0{,}63^{1{,}05} \cdot 1 = 1{,}477. \tag{2}$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 1,5 людино-місяці.

Нижче наведені розрахунки вартості розробки «Платформа дослыдження стохастичних та стохастико-детермынованих алгоритмыв сортування». Основними статтями витрат прийняті:

– основна заробітна плата;
– відрахування на соціальні потреби;
– накладні витрати;
– витрати на персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного продукту і визначається виходячи з кількості розробників, часу виконання розробки (годин), а також заробітної плати в розрахунку на одну годину. Розрахунок заробітної платні проводиться по формі табл. 5.1[3].

Середня заробітна плата залежить від міста. В Києві 30 тисяч гривень, в Дніпрі – 21, в Запоріжжі – 34, Львів – 25, Одесса – 20, Харків – 20 тисяч гривень. Підсумувавши та поділивши, середня зарплатня становить 25000 грн.

Таблиця 5.1 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	25000	1	1,5	37500

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 09.10.2021 до 18.11.2021, що складає 40 дні або 8 робочих тижнів. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка кваліфікованого інженера–програміста складає 148,80 грн/год. Таким чином, витрачено робочого часу:

$$t_{\text{розробки}} = N_{\text{чол}} \cdot N_{\text{тиж}} \cdot N_{\text{год}}, \quad (3)$$

де $N_{\text{чол}}$ – кількість виконавців, *чол*;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, год;

$$t_{\text{розробки}} = 1 \cdot 8 \cdot 40 = 320 \text{чол/год}. \quad (4)$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5)$$

Де $t_{\text{розробки}}$ – витрати праці у чол/год*;*

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, приймається 0,75.

ОЗП складає:

$$ОЗП = 320 \cdot 148{,}80 \cdot 0{,}75 = 35712 \text{грн.} \quad (6)$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати:

$$C_{соц} = \frac{ОЗП \cdot 22\%}{100\%},$$

$$C_{соц} = \frac{35712 \cdot 22\%}{100\%} = 7856 \text{грн.} \quad (7)$$

Отримані результати за (6) та (7) підсумовуються. Вони складають 43 568 грн. та визначають основні прямі витрати.

Накладні витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в процентах (30 – 40%) від суми прямих витрат:

$$C_{накл} = \frac{(ОЗП + C_{соц}) \cdot 35\%}{100\%}; \quad (8)$$

$$C_{накл} = \frac{43568 \cdot 35\%}{100\%} = 15250 \text{грн.} \quad (9)$$

На протязі усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на персональний комп’ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп’ютеру. В експлуатаційні витрати входять:

– вартість витратних матеріалів;
– витрати на ремонт;
– заробітна плата ремонтника;
– оренда приміщення;
– додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;
– амортизаційні витрати на персональний комп'ютер і програмне забезпечення;

– витрати на електроенергію ($С_{ел}$),

які визначаються за формулою:

$$С_{ел} = P \cdot B \cdot T_{розр}, \quad (10)$$

де Р – потужність комп'ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/година, складає 1,44 грн[4];

$T_{розр}$ – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{ел} = 0{,}45 \cdot 1{,}8 \cdot 320 = 207{,}36 грн. \quad (11)$$

Витрати на витратні матеріали ($С_{вм}$) протягом всього терміну експлуатації приблизно 10% від вартості комп'ютеру. Вартість робочої станції приймається 26 000 грн.[5], термін експлуатації – 5 років. Отже, можна визначити ці витрати за період створення програмного засобу:

$$С_{вм} = В_{ком} \cdot \frac{N_Д}{N_{експ} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (12)$$

де $B_{КОМ}$ – вартість персонального комп'ютеру;

$N_Д$ – кількість днів розробки програмного продукту;

$N_{експ}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

$$C_{вм} = 26000 \cdot \frac{40}{5 \cdot 365} \cdot \frac{10}{100} = 57 грн. \quad (13)$$

Заробітна плата ремонтника ($C_{рем}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата починається з 6500 до 12500. Приймається середньою 9000 грн[6]. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$C_{рем} = \frac{C_{рем}'}{N_{КОМ}} \cdot T_{міс}, \quad (14)$$

де $C_{рем}'$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{мес}$ – час розробки програмного продукту, міс.

Заробітна плата ремонтника ($C_{\text{рем}}$) буде складати:

$$C_{\text{рем}} = \frac{9000}{50} \cdot 2 = 360 \text{ грн.}$$

За статистикою витрати на комплектуючи вироби ($\text{С}_{КОМ}$) для ремонту персонального комп'ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{КОМ} = C_{ВМ} = 57 \text{ грн.} \quad (16)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$\text{АКП} = \text{В}_{КОМ} \cdot \frac{N_{\text{Д}}}{N_{\text{експ}} \cdot 365}; \quad (17)$$

$$\text{АКП} = 26000 \cdot \frac{2}{3 \cdot 12} = 1444{,}44$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для Windows 5 років та Intellij IDEA за 1 рік то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп'ютера використовувалася операційна система Windows 10, для написання програмного забезпечення - програмне середовище Intellij IDEA Ultimate.

$$\text{АКП}_w = 13800 \cdot \frac{2}{5 \cdot 12} = 460,$$

$$\text{АКП}_w = 13446 \cdot \frac{2}{12} = 2241.$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2.

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, *грн*	Джерело придбання	Амортизаційні відрахування, *грн*
Windows 10 Professional	13800	http://mtsoft.kiev.ua/product/windows-10-professional	460
Intellij IDEA Ultimate	13446	https://www.jetbrains.com/ru-ru/idea/buy/#commercial	2241
Всього:	38800	-	2701

Додаткові витрати ($C_{\text{дод}}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера-програміст, тобто 12500 гривень на місяць.

Оренду приміщень приймемо рівною 3500 гривень на місяць. На даний момент це найменша ціна на оренду невеликих приміщень у прийтнятному стані, і є поширеною.

Сумарні експлуатаційні витрати на один персональний комп’ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (18)$$

$$C_{\text{експ}} = 207{,}36 + 57 + 360 + 57 + 1444{,}44 + 2701 + 7000 + 12500 = \\ = 24326{,}8\text{грн}. \quad (19)$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	207,36
Вартість витратних матеріалів	57
Витрати на ремонт	360
Амортизація персонального комп’ютера	1444,44
Амортизація програмного забезпечення	2701
Оренда приміщення	7000
Додаткові витрати	12500
Всього	24326,8

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (20)$$

$$C_{\text{розробки}} = 35712 + 7856 + 15250 + 24326 = \\ = 83144 \text{ грн} \quad (21)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	35712,00
Відрахування на соціальні потреби	7856,00
Накладні витрати	15250,00
Експлуатаційні витрати	24326,00
Всього	83144,00

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку сучасного програмного забезпечення для оцінки швидкодії алгоритмів сортування. За результатами розрахунків, приблизна вартість розробки складає 83 144грн.

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

В таблиці табл. 6.1 приведені стадії та етапи розробки програмного забезпечення.

Таблиця 1. Стадії та етапи розробки

№/ з. п.	Зміст робіт	Строки виконання
1	Постановка задачі Узгодження і затвердження ТЗ	15.09.2020 – 30.09.2021
2	Програмування і відладка програми.	01.10.2020 – 15.10.2021
3	Тестування програми	15.10.2021 – 22.10.2021
4	Розробка, узгодження і затвердження програмних документів.	23.10.2021 – 20.11.2021

7. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Для контролю за розробкою продукту кожні 7 днів будуть проводитися зустрічі з замовником для проміжної демонстрації реалізованих функціональних можливостей, обговорення деталей та побажань, а також та за необхідності корегування плану розробки з урахуванням потреб.

Контроль за виконанням роботи здійснює керівник розробки проф. Шинкаренко В.І.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

2. Методики оценки трудозатрат по разработке программного обеспечения информационных систем [Електронне джерело] URL — http://repository.enu.kz/bitstream/handle/data/12881/METODIKA-TRUDOZATRAT.pdf

3. Середня заробітна плата програміста — https://www.work.ua/ru/salary-web%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82/

4. Тарифи на електроенергію — https://index.minfin.com.ua/tariff/electric/

5. Ціна Asus X756U — https://www.moyo.ua/noutbuk_asus_x756uq-t4205d_90nb0c31-m03300/343219.html

6. Середня заробітна плата ремонтника комп’ютерів — https://www.work.ua/ru/salary-%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B9+%D0%BC%D0%B0%D1%81%D1%82%D0%B5%D1%80/

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Борис Боднар

ПЛАТФОРМА ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО-ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01198-01-ЛЗ

Завідувач кафедри КІТ
доц. Вадим Горячкін

Керівник розробки
проф. Віктор Шинкаренко

Виконавець
студент групи ПЗ2021
Костянтин Галанін

Нормоконтролер
доц. Олена Куроп'ятник

2021

ЗАТВЕРДЖЕНО

01116130.1198-01-ЛЗ

ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Специфікація

01116130.95105-01

Листів 2

2021

Позначення	Найменування	Примітка
01116130.01198-01-ЛЗ	Лист затвердження	
01116130.01198-01 12 01-ЛЗ	Лист затвердження	
01116130.01198-01 12 01	Текст програми	
01116130.01198-01 13 01-ЛЗ	Лист затвердження	
01116130.01198-01 13 01	Опис програми	
01116130.01198-01 І3 01	Керівництво користувача. Керівництво з проведення експериментів	

ЗАТВЕРДЖЕНО
01116130.01198-01 13 01

ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Опис програми

01116130.01198-01 13 01

Аркушів 13

2021

АНОТАЦІЯ

Документ 01116130.95105-01 13 01 «Платформа для дослідження стохастичних та стохастико детермінованих алгоритмів сортування. Опис програми» входить до складу програмної документації на програму для реалізації клієнтської частини.

У даному документі представлений опис програми: функціональне призначення, опис логічної структури, виклик і завантаження, вхідні і вихідні дані, порядок роботи з програмою. Програма написані мовою Java. Об'єм пам'яті, що займають програма, складає 36,5 Кб. Конфігурація комп'ютера стандартна.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Програма представляє окремий користувацький додаток та призначена для реалізації інтерфейсу та маніпуляції з ним. За допомогою даної програми користувачі можуть досліджувати задані алгоритми сортування та різні методи їх взаємодії, зокрема використання стохастики для сортування, або предобробки масиву для подальшого сортування.

Для функціонування даного програмного продукту необхідно, щоб на користувацькому комп‘ютері було встановлено наступне програмне забезпечення:

- ОС — Microsoft Windows XP або пізніша, MacOS або інша Linux подібна система;
- Середовище розробки з підтримкою мови Java.

Програмний додаток розробляється з використанням мови програмування Java.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Задачі, що розв‘язуються даним програмним забезпеченням, пов‘язані з отриманням кількісних характеристик виконання алгоритмів сортування. Інформація має бути подана структуровано та зрозуміло.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1. Алгоритм програми

На рис. 1 представлено алгоритм роботи програми.

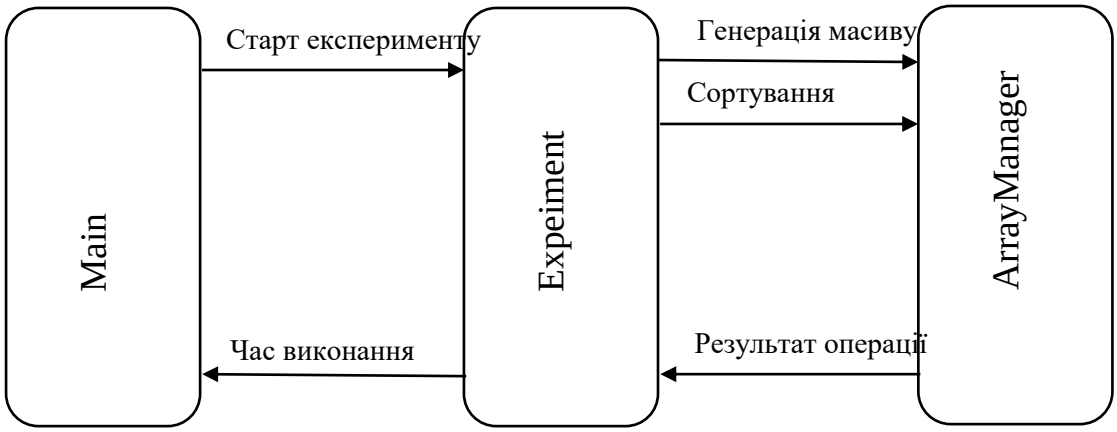

Рисунок 3.1 — алгоритм роботи програми

3.2. Структура програми

На рисунку 2 відображені основні програмні модулі та взаємозв‘язок між ними.

Рисунок 3.2 — взаємозв‘язок програмних модулів

Нижче представлений опис представлених програмних модулів.

1. «Main.class» — модуль має атрибути та реалізує методи, що відповідають за роботу з вхідними даними.
2. «Experiment.class» — модуль має атрибути та реалізує методи, що відповідають за роботу з алгоритмами, замір часу, та підготування сортуємого масиву.
3. «ArrayManager.class» — модуль має атрибути та реалізує методи, що відповідають за роботу роботу алгоритмів сортування.

3.3. Взаємозв’язок з іншими програмами

Додаток функціонує у середовищі розробки, у якому й відображае результати проведення експерименту. Тому для взаємодіі на комп’ютері мають бути встановлені операційна система, рекомендується Windows 10, та середовище розробки з підтримкою мови програмування Java від 5 версії, рекомендується Intelij IDEA.

З іншими програмами даний додаток не взаємодіє та для використання усього функціоналу ні яких інших програм не треба.

4. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для функціонування програмного забезпечення достатньо мати робочий ПК, що має наступні характеристики:

- маніпулюючий пристрій миша;
- клавіатура;
- оперативна пам’ять від 4 ГБ;
- процесор;
- пам’ять достатню для функціонування середовища розробки.

5. ВИКЛИК ТА ЗАВАНТАЖЕННЯ

Програма викликається шляхом натискання лівою кнопкою миші значка виконання файлу в середовищі розробки, відкривши проект у папці «E://SortSearch/src/Main.java» відповідно до того шляху, де було збережено програму.

6. ВХІДНІ ТА ВИХІДНІ ДАНІ

Під даними у даному випадку розуміються символьні значення, які можуть як подаватися користувачем до програми для отримання певних результатів, так і програмою до користувача у якості відповіді на його запит.

6.1. Вхідні дані

Вхідними даними програмного додатку є:

- початковий масив для сортування;
- розмір початкового масиву;
- максимальне та мінімальне значення в масиві;
- алгоритм сортування
- кількість виконання стохастичного алгоритму сортування.

6.2. Вихідні дані:

Вхідними даними програмного додатку є:

- відсортований масив;
- час виконання сортування заданим алгоритмом;
- кількість замін стохастичним алгоритмом сортування.

7. ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

Після запуску програми (E://SortSearch/src/Main.java) на екрані з‘являється головна сторінка проекту (рис. 3).

Головна сторінка складається з панелі взаємодії з проектом, редактору програмного коду, та командного рядка (рис 3).

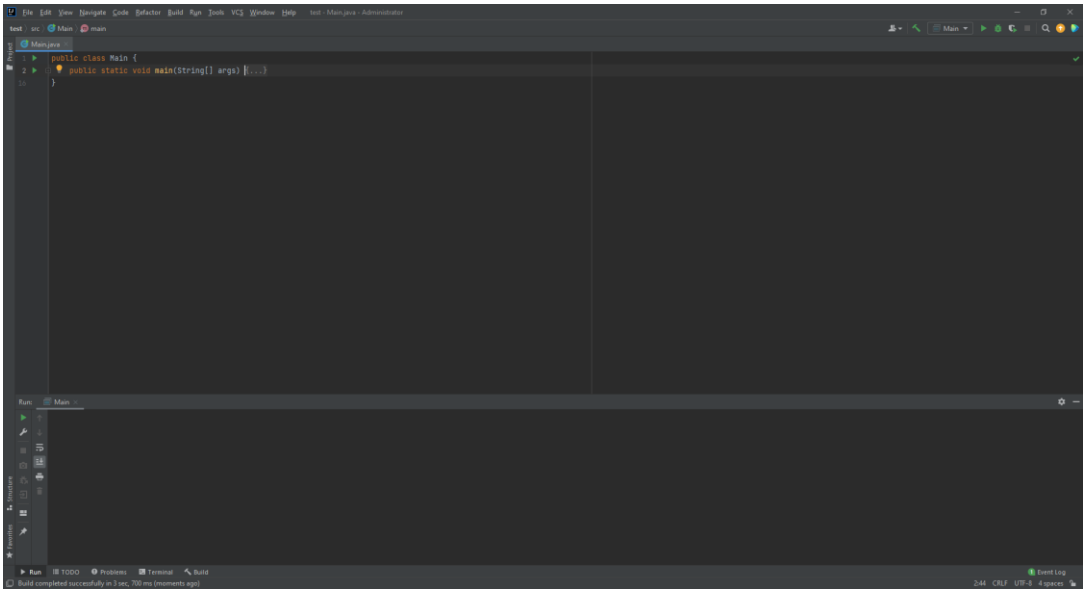

Рисунок 7.1 — головна сторінка програми у середовищі Intellij IDEA

Для внесення зміни алгоритму чи параметрів сортуємого масиву клієнт повинен змінити параметри на 3 – 5 рядках, або функцію, викликану до об’єкту класу Experiment на ту, який алгоритм він хоче дослідити.

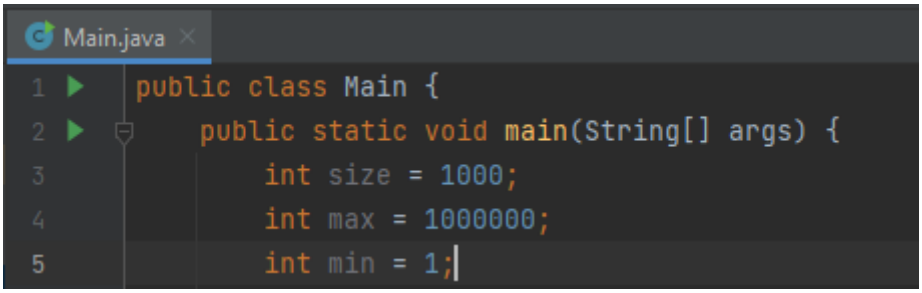

Рисунок 7.2— параметри вхідного масиву

8. ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Програма спроектована таким чином, щоб користувачі могли досліджувати алгоритми не виконуючи їх реалізацію. Для цього реалізовано окремі методи для дослідження взаємодії різних алгоритмів та використання стохастики.

Для виконання першого дослідження достатньо запустити програму у середовищі розробки та натиснути «Запуск».

Для зміни параметрів експерименту перед запуском відкривається головний файл Mein, де усе й задається. Після чого треба натиснути пуск, та у командному рядку буде виведено результати дослідження.

9. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. –38 с.

ЗАТВЕРДЖЕНО

01116130.01198-01-ЛЗ

ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Керівництво користувача

Керівництво з проводження експерименту

01116130.01198-01 ІЗ 01

Аркушів 8

2021

АНОТАЦІЯ

Документ 1116130.01193-01 І3 01 «Платформа для дослідження стохастичних та стохастико-детермінованих алгоритмів сортування. Керівництво користувача входить до складу програмної документації до кваліфікаційного проекту. Даний документ містить опис призначення та умов застосування програми, опис операцій.

ЗМІСТ

ВСТУП

Програмний продукт «Платформа дослідження стохастичних та стохастико-детермінованих алгоритмів сортування» призначений для дослідження ефективності стохастичних алгоритмів сортування та їх використання в детермінованих алгоритмах сортування для покращення їх швидкодії. Ефективність алгоритмів оцінюється часом роботи відносно одне одного.

На сьогоднішній день існує десятки ефективних алгоритмів сортування, які вирішують конкретні задачі за конкретних умов. Але досі не було розроблено ні одного алгоритму на основі стохастики. Використання такого може значно пришвидшити їх виконання, що є затребувано. Тому питання розробки нового, більш ефективного алгоритму є актуальним.

Основною задачею даного програмного продукту є визначення часу роботи вищезгаданих алгоритмів в залежності від вхідних параметрів, обраних алгоритмів, та способів їх комбінування.

1. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Функціонал програми

Програма повинна:

- надавати можливість ввести параметри експерименту;
- сортувати заданий масив;
- відображати часові результати експерименту у консолі

Вхідні дані:

- розмір масиву;
- мінімальне та максимальне значення масиву;
- кількість тестів;
- алгоритм сортування.

1.2 Вимоги до складу і параметрів технічних засобів

Для коректної роботи програми програма повинна виконуватись у середовищі з підтримкою мови Java версії 5 та вище, незалежно від операційної системи, на комп’ютерах, що мають такі мінімальні характеристики:

- 2-ядерний процесор з тактовою частотою не менше 1,6 ГГц;
- 4 ГБ доступної оперативної пам’яті;
- жорсткий диск об’ємом від 128 ГБ.

1.3 Вимоги до інформаційної сумісності:

Вимога до інформаційної та програмної сумісності єдина – програму треба запускати в середовищі розробки з підтримкою мови Java версії 5.

2. ПІДГОТОВКА ДО РОБОТИ

Для роботи з даним програмним продуктом необхідно мати:

- персональний комп’ютер;
- встановлену операційну систему;
- встановлене середовище розробки
- пристрої для взаємодії з комп’ютером – монітор, миша та клавіатура.

2.1 Запуск програмного додатку

Для запуску додатку потрібно виконати наступні кроки у визначеному порядку:

- перейти до папки, яка містить файл проекту Main.java;
- відкрити його у середовищі розробки з підтримкою Java версії 5;
- натиснути на зелений трикутник у правому верхньому куті.

3. ОПИС ОПЕРАЦІЙ

Відкрийте додаток у середовищі розробки та перейдіть до головного файлу Main. На рис 3.1 наведено структуру проекту. На рисунку 3.2 наведено вміст основного файлу.

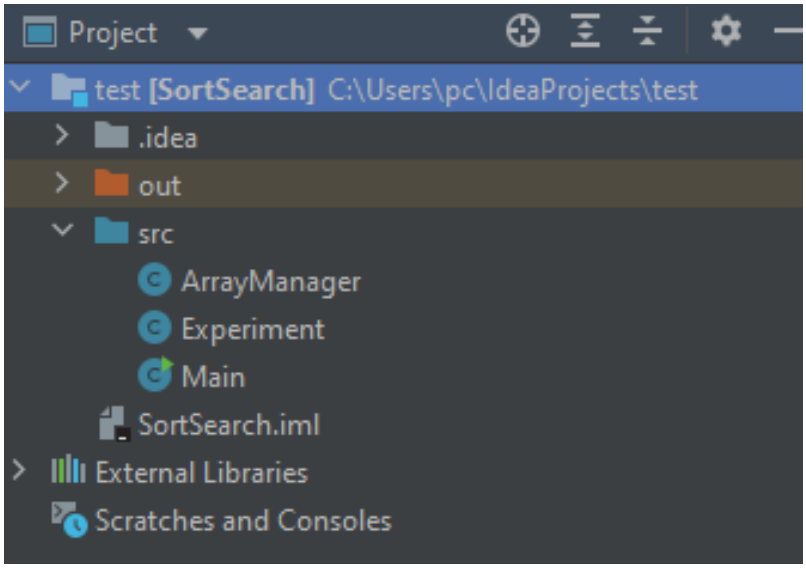

Рисунок 3.1 — Структура проекту

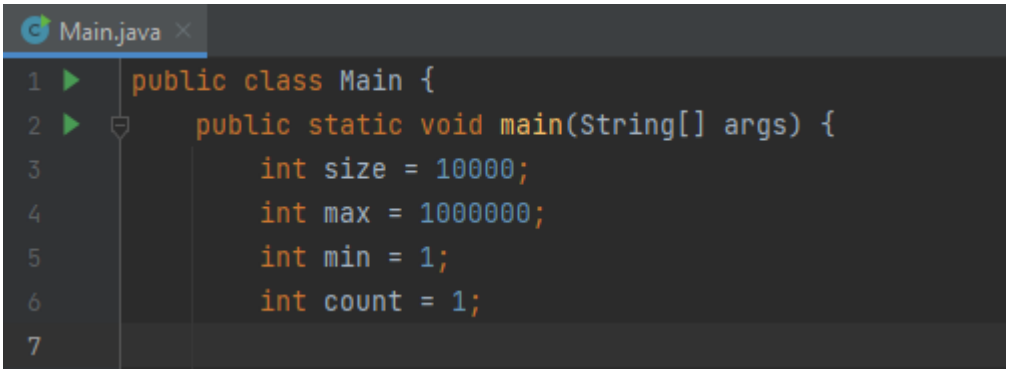

Рисунок 3.2 — Вміст основного файлу

В поля «size», «max» та «min» введіть розмір масиву, максимальне та мінімальне значення. Після цього в поле «count» введіть бажану кількість тестів, які потрібно провести у цьому випробуванні. Потім оберіть один з запропонованих алгоритмів сортування, та впишіть його назву в 14 рядку, замість тексту «Оберіть алгоритм сортування». Переконайтеся, що всі поля правильно заповнені, та середовище не показує помилок та натисніть кнопку «Run».

Після натискання на кнопку на екрані з’явиться командний рядок, який відобразить часові характеристики виконання обраного алгоритму.

Якщо вам потрібно відсортувати ваш конкретний масив даних, введіть в поле «count» один. В 13 рядку замість «size, max, min» додайте посилання на масив даних та вкажіть в 14 рядку потрібний алгоритм сортування. Переконайтесь, що усе заповнено правильно та натисніть кнопку «Run».

Після натискання на екрані з’явиться командний рядок та відобразить час сортування заданого масиву обраним алгоритмом. Після відображення масив відсортовано. На рис 3.3 наведено результат виконання програми.

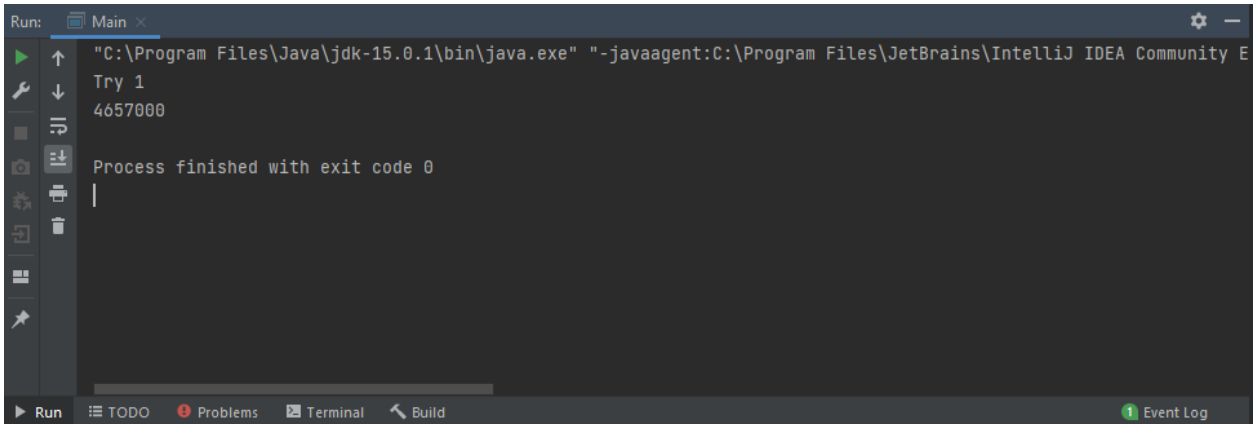

Рисунок 3.3 — Результат виконання програмного додатку.

ЗАТВЕРДЖЕНО
01116130. 01198-01 12 01

ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Текст програми

01116130. 01198-01 12 01

Аркушів 29

2021

АНОТАЦІЯ

Документ 01116130.95105-01 12 01 «платформа для дослідження стохастичних та стохастико детермінованих алгоритмів сортування. Текст програми» входить до складу програмної документації на програму для реалізації клієнтської частини.

У даному документі представлений текст програми клієнтської частини системи. Програми написані мовою Java. Об'єм пам'яті, що займають програми комплексу, складає 36, Кб. Конфігурація комп'ютера стандартна.

ЗМІСТ

1. СХЕМА ВЗАЄМОДІЇ МОДУЛІВ

На рис 1 представлено схему взаємодії модулів.

Рисунок 1.1 — схема взаємодії модулів

2. ТЕКСТ ПРОГРАМИ

2.1 Модуль 1 "Main.class"

/*Модуль має атрибути та реалізує методи, що відповідають за роботу з вхідними даними.*/

//Виконав: студент групи 951м Галанін К. К.

```
public class Main {
    public static void main(String[] args) {
        int size = 1000;
        int max = 1000000;
        int min = 1;
        int count = 1;

        long result = 0;
        Experiment experiment;

        for (int i = 0; i<count; i++){
            System.out.println("Try "+ (i+1));
            experiment = new Experiment(size, max, min);
            //result += experiment. Обрати алгоритм сортування;
        }
        result /=count;
        System.out.println(result);
    }
}
```

2.2 Модуль 2 "Experiment.class"

/* Модуль має атрибути та реалізує методи, що відповідають за роботу з алгоритмами, замір часу, та підготування сортуємого масиву.*/

//Виконав: студент групи 951м Галанін К. К.

```
public class Experiment {
    private static ArrayManager manager = new ArrayManager();
    private boolean toSort = false;

    public Experiment(int size, int max, int min) {
        manager.create(size, max, min);
    }
    public Experiment(int[] arr) {
        manager.create(arr.length, 1, 1);
        toSort = true;
        manager.forSort(arr);
    }

    public long shuffle() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.shuffle();
        time = System.nanoTime() - time;
        return time;
    }
    // стохастичне сортування
    public long shuffleRec() {
        if(!toSort) manager.generate();
```

```
        long time = System.nanoTime();
        manager.shuffleRec();
        time = System.nanoTime() - time;
        return time;
    }

    public long bubble() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.bubbleSort();
        time = System.nanoTime() - time;
        return time;
    }
    // сортування міхуром
    public long shuffleBubble() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.shuffle();
        manager.bubbleSort();
        time = System.nanoTime() - time;
        return time;
    }

    public long tim() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.timSort(manager.arr,
manager.arr.length - 1);
        time = System.nanoTime() - time;
        return time;
    }
    // сортування Тіма
    public long shuffleTim() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.shuffle();
        manager.timSort(manager.arr,
manager.arr.length - 1);
        time = System.nanoTime() - time;
        return time;
    }

    public long smooth() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.smooth_sort(manager.arr, 0,
manager.arr.length - 1);
        time = System.nanoTime() - time;
        return time;
    }
    // плавне сортування
    public long shuffleSmooth() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
        manager.shuffle();
        manager.smooth_sort(manager.arr, 0,
manager.arr.length - 1);
        time = System.nanoTime() - time;
        return time;
    }

    public long quick() {
        if(!toSort) manager.generate();
        long time = System.nanoTime();
```

```
            manager.quickSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }
        // швидке сортування
        public long shuffleQuick() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.shuffle();
            manager.quickSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }

        public long insert() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.insertionSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }
        // сортування вставками
        public long shuffleInsert() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.shuffle();
            manager.insertionSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }

        public long selection() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.selectionSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }
        // сортування вибором
        public long shuffleSelection() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.shuffle();
            manager.selectionSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }

        public long merge() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.mergeSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }
        // сортування злиттям
        public long shuffleMerge() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
```

```
            manager.shuffle();
            manager.mergeSort(manager.arr, 0,
manager.arr.length - 1);
            time = System.nanoTime() - time;
            return time;
        }

        public long heap() {
            if(!toSort)  manager.generate();
            long time = System.nanoTime();
            manager.heapSort();
            time = System.nanoTime() - time;
            return time;
        }
        // сортування злиттям
        public long shuffleHeap() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.shuffle();
            manager.heapSort();
            time = System.nanoTime() - time;
            return time;
        }

        public long shell() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.sort_shell(manager.arr);
            time = System.nanoTime() - time;
            return time;
        }
        // сортування Шелла
        public long shuffleShell() {
            if(!toSort) manager.generate();
            long time = System.nanoTime();
            manager.shuffle();
            manager.sort_shell(manager.arr);
            time = System.nanoTime() - time;
            return time;
        }
    }
```

2.3 Модуль 3 "ArrayManager.class"

/* Модуль має атрибути та реалізує методи, що відповідають за роботу алгоритмів сортування.*/

//Виконав: студент групи 951м Галанін К. К.

```
import java.util.Arrays;


public class ArrayManager {


//    public int counter = 0;
    public int[] arr;
    private int size;
    private int max;
    private int min;
    static int MIN_MERGE = 32;


    public void forSort(int [] arr) {
        this.arr = arr;
        size = arr.length;
    }


    public void clear() {
        size = 10000000;
        min = 1000000;
        max = 10000000;
        double[] dArr = new double[size];
        for (int i = 0; i < size; i++)
            dArr[i] = (min + ((max - min) * Math.random()));
    }

    public void generate() {
        arr = new int[size];
        for (int i = 0; i < size; i++)
            arr[i] = (int) (min + ((max - min) * Math.random()));
    }


    public void swap(int[] array, int first, int second) {


        int tmp = array[first];
        array[first] = array[second];
        array[second] = tmp;
    }


    public void shuffleRec() {
        boolean state = false;
        while (!state) {
            boolean pState = true;
            for (int i = 0; i < Math.sqrt(size); i++) {
                int first = (int) (Math.random() * arr.length);
                int second = (int) (Math.random() * arr.length);
                if (first < second) {
                    if (arr[first] > arr[second]) {
//                        counter++;
```

```
                swap(arr, first, second);
                pState = false;
            }
        } else if (first == second) {
            continue;
        } else {
            if (arr[second] > arr[first]) {
//                counter++;
                swap(arr, first, second);
                pState = false;
            }
        }
        state = pState;
    }
//    System.out.println("Swaps: " + counter);
//    counter = 0;
  }

  public void shuffle() {
    for (int i = 0; i < Math.sqrt(size); i++) {
      int first = (int) (Math.random() * arr.length);
      int second = (int) (Math.random() * arr.length);
      if (first < second) {
        if (arr[first] > arr[second]) {
          swap(arr, first, second);
        }
      } else if (first == second) {
        continue;
      } else {
        if (arr[second] > arr[first]) {
          swap(arr, first, second);
        }
      }
    }
  }

  public void bubbleSort() {
    boolean isSorted = false;
    int tmp;
    while (!isSorted) {
      isSorted = true;
      for (int i = 0; i < arr.length - 1; i++) {
        if (arr[i] > arr[i + 1]) {
          isSorted = false;

          tmp = arr[i];
          arr[i] = arr[i + 1];
          arr[i + 1] = tmp;
        }
      }
    }
  }

  public void quickSort(int[] array, int low, int high) {
    if (array.length == 0)
      return;

    if (low >= high)
      return;

    int middle = low + (high - low) / 2;
```

```
    int opora = array[middle];

    int i = low, j = high;
    while (i <= j) {
      while (array[i] < opora) {
        i++;
      }

      while (array[j] > opora) {
        j--;
      }

      if (i <= j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        i++;
        j--;
      }
    }

    if (low < j)
      quickSort(array, low, j);

    if (high > i)
      quickSort(array, i, high);
  }

  public void mergeSort(int[] array, int left, int right)
{
    if (right <= left) return;
    int mid = (left + right) / 2;
    mergeSort(array, left, mid);
    mergeSort(array, mid + 1, right);
    merge(array, left, mid, right);
  }

  void merge(int[] array, int left, int mid, int right) {
    int lengthLeft = mid - left + 1;
    int lengthRight = right - mid;

    int leftArray[] = new int[lengthLeft];
    int rightArray[] = new int[lengthRight];

    for (int i = 0; i < lengthLeft; i++)
      leftArray[i] = array[left + i];
    for (int i = 0; i < lengthRight; i++)
      rightArray[i] = array[mid + i + 1];

    int leftIndex = 0;
    int rightIndex = 0;

    for (int i = left; i < right + 1; i++) {
      if (leftIndex < lengthLeft && rightIndex <
lengthRight) {
        if (leftArray[leftIndex] <
rightArray[rightIndex]) {
          array[i] = leftArray[leftIndex];
          leftIndex++;
        } else {
          array[i] = rightArray[rightIndex];
          rightIndex++;
        }
      } else if (leftIndex < lengthLeft) {
```

```
            array[i] = leftArray[leftIndex];
            leftIndex++;
        } else if (rightIndex < lengthRight) {
            array[i] = rightArray[rightIndex];
            rightIndex++;
        }
    }
}

void heapify(int[] array, int length, int i) {
    int leftChild = 2 * i + 1;
    int rightChild = 2 * i + 2;
    int largest = i;

    if (leftChild < length && array[leftChild] > array[largest]) {
        largest = leftChild;
    }

    if (rightChild < length && array[rightChild] > array[largest]) {
        largest = rightChild;
    }

    if (largest != i) {
        int temp = array[i];
        array[i] = array[largest];
        array[largest] = temp;
        heapify(array, length, largest);
    }
}

public void heapSort() {
    if (arr.length == 0) return;

    int length = arr.length;
    for (int i = length / 2 - 1; i >= 0; i--)
        heapify(arr, length, i);

    for (int i = length - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

public void insertionSort(int[] arr, int left, int right) {
    for (int i = left + 1; i <= right; i++) {
        int temp = arr[i];
        int j = i - 1;
        while (j >= left && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}

public void selectionSort(int[] numbers, int low, int high) {
    for (int h = low; h <= high; h++)
```

```
        swap(numbers, h, getSmallest(numbers, h, high));
    }

    public int getSmallest(int[] numbers, int low, int high) {
        int small = low;
        for (int i = low + 1; i <= high; i++)
            if (numbers[i] < numbers[small])
                small = i;
        return small;
    }

    public static int minRunLength(int n) {
        assert n >= 0;

        int r = 0;
        while (n >= MIN_MERGE) {
            r |= (n & 1);
            n >>= 1;
        }
        return n + r;
    }

    public void timSort(int[] arr, int n) {
        int minRun = minRunLength(MIN_MERGE);

        for (int i = 0; i < n; i += minRun) {
            insertionSort(arr, i,
                    Math.min((i + MIN_MERGE - 1), (n - 1)));
        }

        for (int size = minRun; size < n; size = 2 * size) {

            for (int left = 0; left < n;
                 left += 2 * size) {

                int mid = left + size - 1;
                int right = Math.min((left + 2 * size - 1),
                        (n - 1));

                if (mid < right)
                    merge(arr, left, mid, right);
            }
        }
    }

    void sort_shell(int[] a) {
        int i, j, k, h, m = 0, b = a.length;
        int[] d = {1, 4, 10, 23, 57, 145, 356, 911, 1968, 4711, 11969, 27901,
                84801, 213331, 543749, 1355339, 3501671, 8810089, 21521774,
                58548857, 157840433, 410151271, 1131376761, 2147483647};
        while (d[m] < b) ++m;
        while (--m >= 0) {
            k = d[m];
            for (i = k; i < b; i++) {
                j = i;
                h = a[i];
                while ((j >= k) && (a[j - k] > h)) {
                    a[j] = a[j - k];
                    j -= k;
                }
```

```
            a[j] = h;
          }
        }
      }

    static final int LP[] = {1, 1, 3, 5, 9, 15, 25, 41, 67, 109,
          177, 287, 465, 753, 1219, 1973, 3193, 5167, 8361, 13529, 21891,
          35421, 57313, 92735, 150049, 242785, 392835, 635621, 1028457,
          1664079, 2692537, 4356617, 7049155, 11405773, 18454929, 29860703,
          48315633, 78176337, 126491971, 204668309, 331160281, 535828591,
          866988873 // the next number is > 31 bits.
    };

    public void smooth_sort(int[] m, int lo, int hi) {
        int head = lo;
        int p = 1;
        int pshift = 1;

        while (head < hi) {
            if ((p & 3) == 3) {
                sift(m, pshift, head);
                p >>>= 2;
                pshift += 2;
            } else {

                if (LP[pshift - 1] >= hi - head) {

                    trinkle(m, p, pshift, head, false);
                } else {
                    sift(m, pshift, head);
                }

                if (pshift == 1) {

                    p <<= 1;
                    pshift--;
                } else {

                    p <<= (pshift - 1);
                    pshift = 1;
                }
            }
            p |= 1;
            head++;
        }

        trinkle(m, p, pshift, head, false);

        while (pshift != 1 || p != 1) {
            if (pshift <= 1) {
                // block of length 1. No fiddling needed
                int trail = Integer.numberOfTrailingZeros(p & ~1);
                p >>>= trail;
                pshift += trail;
            } else {
                p <<= 2;
                p ^= 7;
                pshift -= 2;
```

```
            // This block gets broken into three bits. The rightmost
            // bit is a block of length 1. The left hand part is split into
            // two, a block of length LP[pshift+1] and one of LP[pshift].
            // Both these two are appropriately heapified, but the root
            // nodes are not necessarily in order. We therefore semitrinkle
            // both of them

            trinkle(m, p >>> 1, pshift + 1, head - LP[pshift] - 1, true);
            trinkle(m, p, pshift, head - 1, true);
        }

        head--;
    }
}

private void sift(int[] m, int pshift, int head) {

    int val = m[head];

    while (pshift > 1) {
        int rt = head - 1;
        int lf = head - 1 - LP[pshift - 2];

        if (Integer.compare(val, m[lf]) >= 0 && Integer.compare(val, m[rt]) >= 0)
            break;
        if (Integer.compare(m[lf], m[rt]) >= 0) {
            m[head] = m[lf];
            head = lf;
            pshift -= 1;
        } else {
            m[head] = m[rt];
            head = rt;
            pshift -= 2;
        }
    }

    m[head] = val;
}

private void trinkle(int[] m, int p, int pshift, int head, boolean isTrusty) {

    int val = m[head];

    while (p != 1) {
        int stepson = head - LP[pshift];

        if (Integer.compare(m[stepson], val) <= 0)
            break;
        if (!isTrusty && pshift > 1) {
            int rt = head - 1;
            int lf = head - 1 - LP[pshift - 2];
            if (Integer.compare(m[rt], m[stepson]) >= 0
                    || Integer.compare(m[lf], m[stepson]) >= 0)
                break;
        }

        m[head] = m[stepson];
```

```
            head = stepson;
            int trail = Integer.numberOfTrailingZeros(p &
~1);
            p >>>= trail;
            pshift += trail;
            isTrusty = false;
        }

        if (!isTrusty) {
}
```

```
                m[head] = val;
                sift(m, pshift, head);
            }
        }

        public void create(int size, int max, int min){
            this.size = size;
            this.max = max;
            this.min = min;
        }
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
АТ «УКРАЇНСЬКА ЗАЛІЗНИЦЯ»
ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО
ТРАНСПОРТУ ІМЕНІ АКАДЕМІКА В. ЛАЗАРЯНА
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФРАСТРУКТУРИ ТА ТЕХНОЛОГІЙ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ АВТОМОБІЛЬНО-ДОРОЖНІЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

81 Всеукраїнської науково-технічної конференції
молодих учених, магістрантів та студентів

«НАУКА І СТАЛИЙ РОЗВИТОК ТРАНСПОРТУ»

28 жовтня 2021 року

INFORMATION-TELECOMMUNICATION TECHNOLOGY TA COMPUTER MODELING

CONFERENCE PROCEEDINGS

81th all Ukrainian Scientific and Technical Conference
of young scientists, masters and students

"SCIENCE AND SUSTAINABLE DEVELOPMENT OF TRANSPORT"

October 28, 2021

ДНІПРО
2021

УДК 658.512.2:681.3.06

Інформаційно-телекомунікаційні технології та комп'ютерне моделювання [електронний ресурс]: збірник тез доповідей секції 81 Всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту» 28 жовтня 2021 р. – Дніпро: Дніпровський нац. ун-т залізн. трансп. ім. акад. В. Лазаряна, 2021. – 29 с. – URL: http://ndch.diit.edu.ua/upload/%D0%9A%D0%BE%D0%BD%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B8/2021/81_All_UA_ST_Conference_of_YSMS_SSD_of_Transport/Infotelecom_and_Computer_Simulation_2021.pdf

У збірнику тез доповідей подано результати досліджень здобувачів вищої освіти і молодих учених, які присвячено питанням інформаційно-телекомунікаційних технологій та комп'ютерного моделювання, їх розвитку і поширенню сфер застосування. Тези доповідей подано в рамках 81 Всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту», яку проведено 28 жовтня 2021 року у Дніпровському національному університеті залізничного транспорту імені академіка В. Лазаряна.

Збірник тез доповідей призначено для здобувачів вищої освіти і молодих учених.

Текст тез доповідей учасників конференції подано в авторській редакції.

Офіційна наукова конференція здобувачів вищої освіти та молодих учених:

– Лист Державної наукової установи «Інститут модернізації змісту освіти» від 19.01.2021 р. № 22.1/10-83 «Про Перелік міжнародних, всеукраїнських науково-практичних конференцій здобувачів вищої освіти і молодих учених».

ЗМІСТ

ДОСЛІДЖЕННЯ СТОХАСТИЧНИХ ТА СТОХАСТИКО-ДЕТЕРМІНОВАНИХ АЛГОРИТМІВ СОРТУВАННЯ

Автор: Галанін К. К., студент групи ПЗ 2021
Науковий керівник: д.т.н., професор Шинкаренко В. І.
Дніпровський національний університет залізничного транспорту
імені академіка В. Лазаряна

У сучасному інформаційному просторі об'єм інформації збільшується в геометричній послідовності. Та для її обробки необхідно насамперед її сортувати.

Зараз існують десятки різних алгоритмів сортування, кожен з яких краще працює в окремому випадку, хто краще працює з великими масивами даних, хто краще за швидкістю, або за кількістю процесорних циклів.

Переваги стохастичних алгоритмів сортування:

- регулювання кількості операцій;
- можна використовувати разом з іншими алгоритмами або замість них;
- не мають залежностей від розмірів массиву та початкового порядку элементів.

Метою роботи є дослідження роботи стохастичних та стохастико-детермінованих алгоритмів сортування.

Визначено переваги використання стохастичних алгоритмів для сортування різних обсягів даних.

В порівнянні з іншими алгоритмами сортування стохастичні алгоритми більш гнучкі, та можуть корисно використовуватися разом з іншими детермінованими алгоритмами для покращення показників їх швидкодії.

Виконано аналіз існуючих алгоритмів сортування за результатами якого визначено, що функціональність існуючих алгоритмів дуже обмежена.

Здійснено експерименти з використання алгоритмів у задачах різних типів, за результатами яких доведено, що робота існуючих алгоритмів у багатьох випадках не є рентабельною.

Коли потрібне приблизне сортування, або не потрібна його точність, детерміновані алгоритми будуть виконувати багато даремних операцій для завершення сортування. Відповідно буде задіяно більше ресурсів та затрачено більше часу.

При точному сортуванні стохастичними алгоритмами можна зробити початкову обробку масиву, після чого успішно використати значно легші детерміновані алгоритми, які будуть використовувати набагато менше процесорного часу та ресурсів системи.

Реалізовано програмні засоби реалізації стохастичних алгоритмів сортування для різних сценаріїв їх використання.

Виконано аналіз використання стохастичних алгоритмів з/замість існуючими детермінованими алгоритмами сортування.

Запропоновано модель використання стохастичних алгоритмів сортування, та сценарії з найбільшою ефективністю.