МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Розрахункова робота № 1**

з дисципліни « Алгоритми і структури даних »

*назва дисципліни*

на тему: «ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОДУКТИВНОСТІ РОБОТИ СТРУКТУР ДАНИХ ТА ДОСЛІДЖЕННЯ АЛГОРИТМУ СОРТУВАННЯ»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: канд. техн. наук, доцент на кафедрі 603 Волобуєва Ліна Олексіївна

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

Зміст

[Зміст 2](#_Toc153738669)

[Постановка задачі 2](#_Toc153738670)

[Варіант завдання до розрахункової роботи 1: 3](#_Toc153738671)

[Хід роботи 4](#_Toc153738672)

[Опис програми: 4](#_Toc153738673)

[Теоретичні відомості 4](#_Toc153738674)

[Вербальний опис алгоритму сортування М (Обмінне сортування зі злиттям) 4](#_Toc153738675)

[Опис алгоритму сортування М (Обмінне сортування зі злиттям) у вигляді блок схеми 5](#_Toc153738676)

[Приклад, для показу сутності алгоритму сортування чисел з докладним описом кожного кроку 5](#_Toc153738677)

[Лістинг програми 7](#_Toc153738678)

[Екранні форми результатів роботи програми 9](#_Toc153738679)

[Висновок 10](#_Toc153738680)

**Мета роботи** − виконати порівняльний аналіз продуктивності роботи структур даних та дослідження алгоритму сортування

Постановка задачі

1. Виконати порівняльний аналіз продуктивності роботи лінійного списку (лаб. роб. 1) і системного неузагальненого та узагальненої черги в залежності від варіанту. Для цього:
   1. Доопрацювати свою лаб. роб. 1 таким чином, щоб була можливість додавати данні у вигляді структури/ об’єкту типу ключ- значення.
   2. Заміряти час додавання та видалення n-даних (n>10000).
   3. Заміряти час додавання та видалення n-даних (n>10000) для системного неузагальненої черги.
   4. Заміряти час додавання та видалення n-даних (n>10000) для системного узагальненоої черги.
   5. Побудувати графіки залежності часу додавання/ видалення даних від кількості елементів для реалізації лаб. роб. 1, неузагальненого та узагальненої черги. На одній координатній площині додавання для 3-х структур, на другій видалення. Кількість точок для побудови графіка 5-10.
2. Доопрацювати свою лаб. роб. 6 таким чином, щоб була можливість задавати данні у вигляді структури/ об’єкту типу ключ-значення і відсортовувати в залежності від значення ключа. Заміряти час виконання в залежності від числа елементів і початкового виду масиву. Розглянути ситуації:
   1. Елементи в масиві розташовані у випадковому порядку
   2. Елементи в масиві вже відсортовані
   3. Елементи в масиві розташовані у зворотному порядку
   4. Побудувати графіки з даними досліджень.
   5. Побудувати графіки з даними досліджень (з графіка на одній координатній площині). Для побудови графіків можна використовувати Excel.
3. Оформити звіт з наступною структурою:
   1. Титульний аркуш.
   2. Зміст (генерується автоматично).
   3. Постановка завдання (із зазначенням номеру свого варіанту, структурою, що досліджується та назвою алгоритму сортування).
   4. Опис класів необхідних для заміру часу виконання операцій.
   5. Проектування необхідних класів і структур (допустимо для кожної задачі окремо).
   6. Екранні форми роботи програми (допустимо для кожної задачі окремо).

**Варіант 9** завдання до розрахункової роботи 1:

**Структура, що досліджується:** лінійний двонаправлений список символів на черзі.

**Алгоритм сортування:** паралельне сортування Бетчера. Алгоритм M(обміну зі злиттям). (Batcher's Merge-Exchange Sort)

Хід роботи

Опис програми:

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 6), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

Опис класів необхідних для заміру часу виконання операцій.

**Клас TimeTests**

Напівуніверсальний клас для виконання операцій додавання та видалення елементів типу KeyValue з різних типів системних черг (системна узагальнена/неузагальнена черга).

Клас має приватне поле з об’єктом класу System.Diagnostics.Stopwatch для визначення затраченого часу на виконання операцій.

Має метод, який визиває метод додавання елементів з порядковим значенням ключа та випадковим значенням символу (char) об’єкту типу KeyValue та видалення цих же елементів з використанням лічильника часу.

Окремий метод для виклику методів додавання об’єктів KeyValue зі встановленими значеннями ключа з порядковим значенням ключа та випадковим значенням символу (char) об’єкту типу KeyValue та видалення з об’єктів класу власної черги.

Повний код класу подано у додатку А.

**Клас SortTimeTest**

Клас для запуску тестів швидкості виконання сортування різних типів масиву з об’єктами типу KeyValue.

Містить загальний метод, який викликає 3 тести швидкості сортування для кожної заданої кількості елементів.

Модифікований метод алгоритму сортування М, без виведення кроків на консоль.

Метод тестування часу сортування переданого масиву з визначенням часу та виведенням результату на екран.

3 методи, які генерують масиви певного вигляду: псевдовипадковий масив з псевдовипадковими ключами в об’єктах типу KeyValue та псевдовипадковий символ для Value; масив об’єктів типу KeyValue зі значеннями ключів по порядку та псевдовипадковий символ для Value; масив об’єктів типу KeyValue зі значеннями ключів в спадаючому порядку.

Повний код класу подано у додатку Б.

Екранні форми результатів замірів часу виконання процесів додавання, видалення та сортування

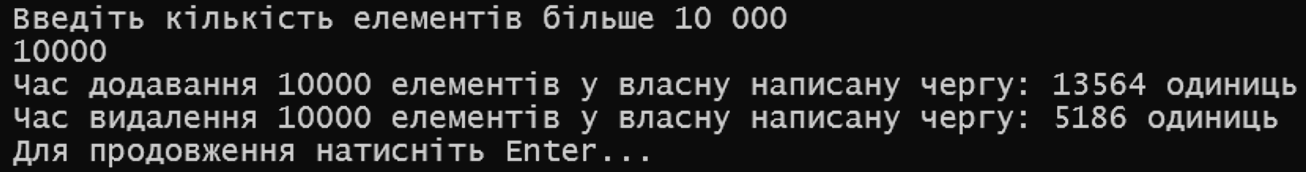


Рисунок 1 – тест швидкості роботи власно написаної черги з 10 тис. елементами

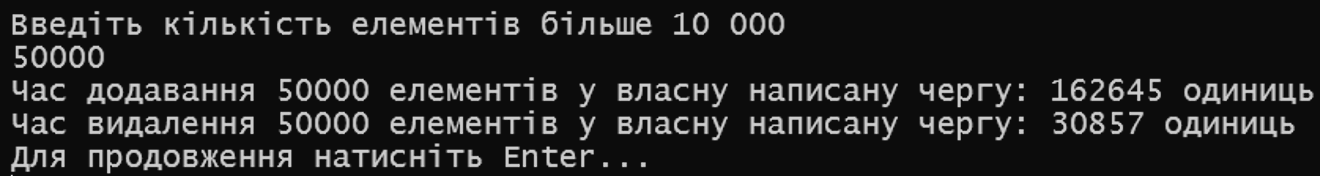


Рисунок 2 – тест швидкості роботи власно написаної черги з 50 тис. елементами

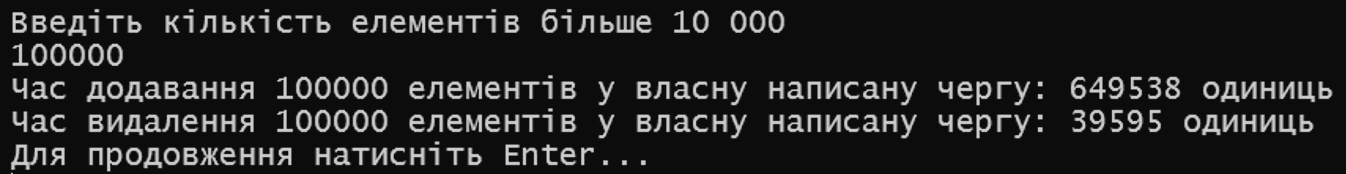


Рисунок 3 – тест швидкості роботи власно написаної черги зі 100 тис. елементами

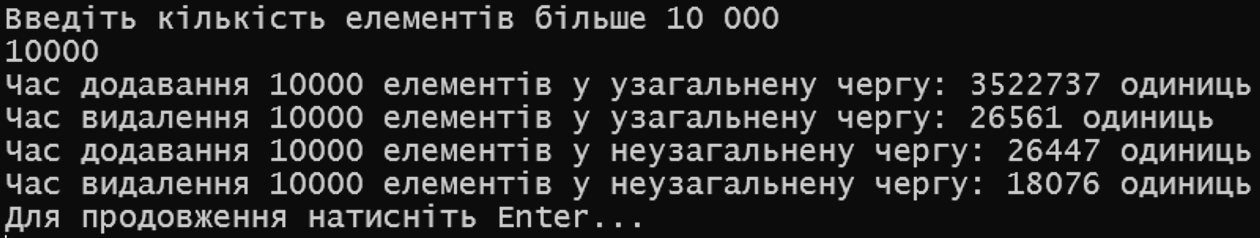


Рисунок 4 – тест швидкості роботи системних черг з 10 тис. елементами

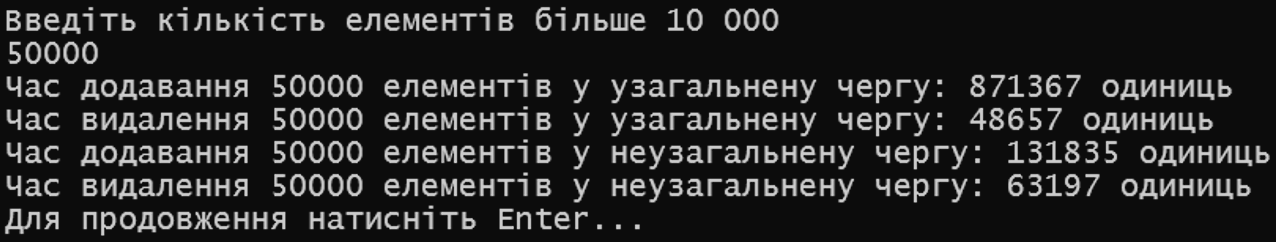


Рисунок 5 – тест швидкості роботи системних черг з 50 тис. елементами

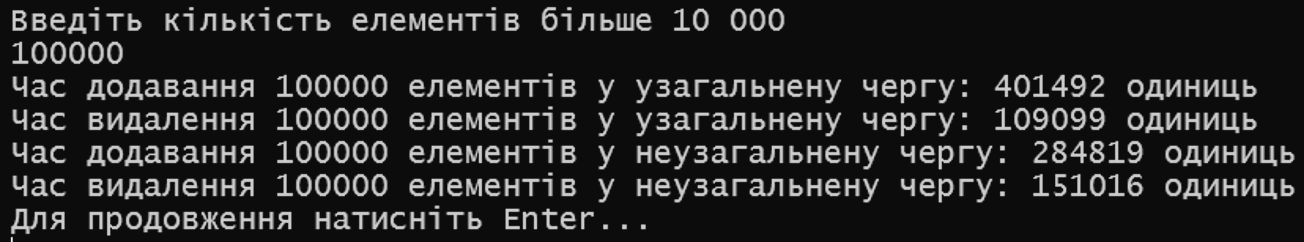


Рисунок 6 – тест швидкості роботи системних черг зі 100 тис. елементами

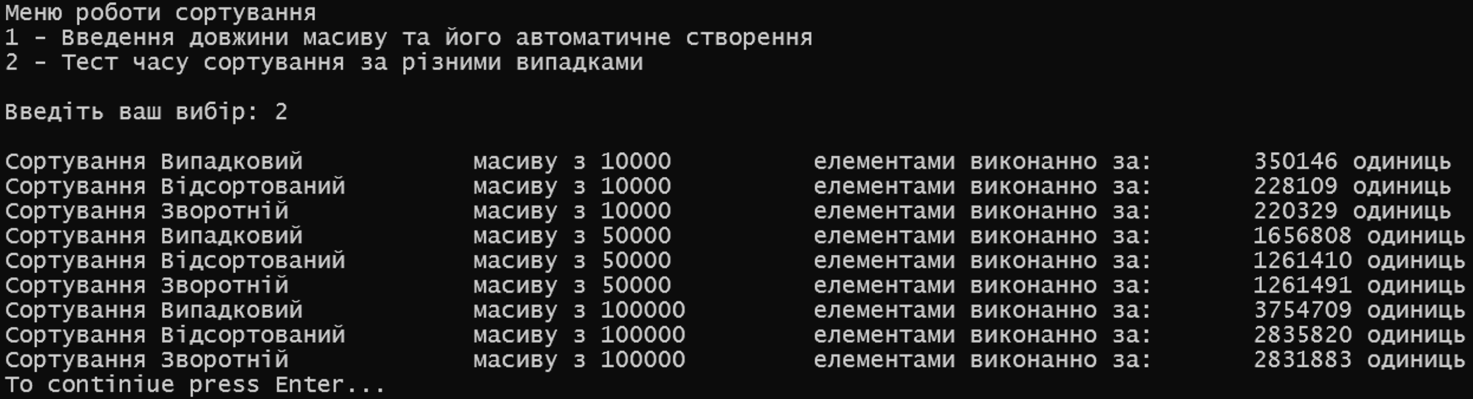


Рисунок 7 – тест швидкості роботи сортування з різними випадками масивів та кількістю елементів

Графіки побудовані за результатами досліджень

Запишемо результати тестування у відповідні таблиці та побудуємо графіки

Таблиця 1 – Залежність швидкості додавання елементів від кількості елементів та типу структури

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № заміру | Кількість елементів | Затрачений час, од. | | |
| Власний клас черги | Системна узагальнена черга | Системна неузагальнена черга |
| 1 | 10 000 | 51799 | 36993 | 22993 |
| 2 | 15613 | 20083 | 26294 |
| 3 | 50 000 | 586180 | 92636 | 92207 |
| 4 | 94982 | 311894 | 160529 |
| 5 | 100 000 | 408859 | 178733 | 160546 |
| 6 | 585621 | 424912 | 135258 |

Таблиця 2 – Залежність швидкості видалення елементів від кількості елементів та типу структури

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № заміру | Кількість елементів | Затрачений час, од. | | |
| Власний клас черги | Системна узагальнена черга | Системна неузагальнена черга |
| 1 | 10 000 | 13570 | 8780 | 7989 |
| 2 | 3215 | 4838 | 12008 |
| 3 | 50 000 | 132706 | 26462 | 39623 |
| 4 | 41051 | 19113 | 43789 |
| 5 | 100 000 | 55270 | 41665 | 75493 |
| 6 | 42022 | 31162 | 71916 |

Таблиця 3 – Залежність швидкості видалення елементів від кількості елементів та типу структури

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № заміру | Кількість елементів | Затрачений час, од. | | |
| Випадковий порядок | Вже відсортовано | Зворотній порядок |
| 1 | 10 000 | 301239 | 200161 | 191792 |
| 2 | 276334 | 181592 | 170351 |
| 3 | 50 000 | 1777222 | 1243713 | 1250146 |
| 4 | 1442161 | 1067728 | 1109614 |
| 5 | 100 000 | 4183835 | 2822698 | 2872916 |
| 6 | 3268229 | 2422996 | 2431061 |

Рисунок 8 – діаграма швидкості додавання елементів

Рисунок 9 – діаграма швидкості видалення елементів

Рисунок 10 – діаграма швидкості сортування елементів

Висновок

Для виконання порівняльного аналізу продуктивності роботи лінійного списку та системного неузагальненого та узагальненої черги, внесено низку модифікацій до моєї лабораторної роботи. Доопрацювавши лаб. роб. 1, я забезпечив можливість додавати дані у вигляді структури/об'єкта типу ключ-значення. Після цього заміряв час додавання та видалення різних кількостей даних (n>10000) для лінійного списку, системного неузагальненого та узагальненої черги.

Дослідження показало, що продуктивність власної черги значно відрізняється від продуктивності системних структур в гіршу сторону.

В роботі також доопрацьовано лаб. роб. 6, щоб надавати дані у вигляді об'єкта типу ключ-значення і відсортовувати їх в залежності від значення ключа. Заміряні були час виконання для різних типів вхідних масивів: у випадковому порядку, вже відсортованому та у зворотному порядку.

Результати досліджень вказують на те, що продуктивність алгоритму сортування значно залежить від вхідного порядку елементів. У випадку масиву у зворотному порядку досягається значний приріст швидкості виконання сортування, майже як у вже відсортованого масиву, на відміну від повністю випадково згенерованого масиву.

Побудовані графіки наглядно демонструють залежність часу виконання операцій від розміру вхідних даних і їхнього початкового порядку. Використання Excel дозволило зручно візуалізувати результати та зробити висновки щодо ефективності різних підходів.

Враховуючи отримані результати, можна визначити оптимальний вибір структури даних та алгоритму сортування в конкретних умовах застосування. Дані експериментів служать підставою для обґрунтування вибору оптимального рішення в конкретних випадках використання структур даних та алгоритмів сортування.

*Додаток A*  
Код реалізації класу для заміру часу у роботі з чергами

namespace ASD\_RR\_1

{

public class TimeTests

{

private static System.Diagnostics.Stopwatch watch = new System.Diagnostics.Stopwatch();

public static void PerformSQueueOperations<T>(T queue, int count)

{

EnqueueEl(queue, count);

DequeueEl(queue, count);

}

private static void EnqueueEl<T>(T queue, int count)

{

watch.Restart();

for (int i = 0; i < count; i++)

{

dynamic dynamicQueue = queue;

dynamicQueue.Enqueue(new KeyValue { Key = i, Value = (char)('A' + i % 26) });

}

watch.Stop();

Console.WriteLine($"Час додавання {count} елементів у {GetQueueTypeName(queue)} чергу: {watch.ElapsedTicks} одиниць");

}

private static void DequeueEl<T>(T queue, int count){

watch.Restart();

for (int i = 0; i < count; i++){

dynamic dynamicQueue = queue;

dynamicQueue.Dequeue();

}

watch.Stop();

Console.WriteLine($"Час видалення {count} елементів у {GetQueueTypeName(queue)} чергу: {watch.ElapsedTicks} одиниць");

}

public static void PerformMyQueueOperations(MyQueue queue, int count){

EnQueueT(queue, count);

DeQueueT(queue, count);}

private static void EnQueueT(MyQueue queue, int count)

{

watch.Restart();

for (int i = 0; i < count; i++)

queue.EnQueue(new KeyValue { Key = i, Value = (char)('A' + i % 26) });

watch.Stop();

Console.WriteLine($"Час додавання {count} елементів у власну написану чергу: {watch.ElapsedTicks} одиниць"); }

private static void DeQueueT(MyQueue queue, int count) {

watch.Restart();

for (int i = 0; i < count; i++)

queue.DeQueue();

watch.Stop();

Console.WriteLine($"Час видалення {count} елементів у власну написану чергу: {watch.ElapsedTicks} одиниць");

}

private static string GetQueueTypeName<T>(T queue)

{ Type type = queue.GetType();

if (type.IsGenericType) return "узагальнену";

else return "неузагальнену";}}

}

*Додаток Б*  
Код реалізації класу для заміру часу сортування

namespace ASD\_RR1{

public class SortTimeTest

{

public static void RunSortingTests()

{

int[] inputCounts = { 10000, 50000, 100000 }; // Різні розміри масивів для тестів

foreach (var count in inputCounts)

{

// Генеруємо масиви для різних сценаріїв

KeyValue[] randomArray = GenerateRandomArray(count);

KeyValue[] orderedArray = GenerateOrderKeyValueArray(count);

KeyValue[] reversedArray = GenerateReversedKeyValueArray(count);

// Тестуємо сортування для різних сценаріївз

TestSorting(randomArray, "Випадковий");

TestSorting(orderedArray, "Відсортований");

TestSorting(reversedArray, "Зворотній");

}

}

private static void TestSorting(KeyValue[] array, string arrayType)

{

// Засікаємо час сортування

System.Diagnostics.Stopwatch watch = new System.Diagnostics.Stopwatch();

watch.Start();

// Викликаємо ваш метод сортування

InsideMSort(array);

watch.Stop();

// Виводимо результат

Console.WriteLine($"Сортування \t{arrayType}\t масиву з \t{array.Length}\t елементами виконанно за:\t{watch.ElapsedTicks} одиниць");

}

static KeyValue[] GenerateRandomArray(int length)

{

Random random = new Random();

KeyValue[] array = new KeyValue[length];

for (int i = 0; i < length; i++)

{

array[i] = new KeyValue

{

Key = random.Next(-1000, 1000), // Генеруємо випадкові числа в діапазоні (-1000, 1000)

Value = (char)random.Next(32, 127) // Генеруємо випадковий символ з ASCII-таблиці

};

}

return array;

}

public static KeyValue[] GenerateOrderKeyValueArray(int input\_count)

{

Random random = new Random();

KeyValue[] array = new KeyValue[input\_count];

for (int i = 0; i < input\_count; i++)

{

array[i] = new KeyValue

{

Key = i,

Value = (char)random.Next(32, 127)

};

}

return array;

}

public static KeyValue[] GenerateReversedKeyValueArray(int input\_count)

{

Random random = new Random();

KeyValue[] array = new KeyValue[input\_count];

for (int i = input\_count - 1; i >= 0; i--)

{

array[i] = new KeyValue

{

Key = i + 1,

Value = (char)random.Next(32, 127)

};

}

return array;

}

public static void InsideMSort(KeyValue[] array)

{

int iteration = 1;

int N = array.Length;

int t = (int)Math.Ceiling(Math.Log2(N));

int p = (int)Math.Pow(2, t - 1); // початкова установка p M1

do {

int q = (int)Math.Pow(2, t - 1);//початкова установка q, r, d M2

int r = 0;

int d = p;

do

{

for (int i = 0; i <= N - d - 1; i++) // цикл по i M3

{

if (i + d < N && i < N - d && array[i].Key > array[i + d].Key && ((i & p) == r)) // порівняння\обмін елементів масиву M4

{

KeyValue temp\_swap = array[i];

array[i] = array[i + d];

array[i + d] = temp\_swap;

}

}

if (q != p)

{

d = q - p;

q /= 2;

r = p;

}

else

{

break; // вихід з циклу, якщо q=p

}

} while (q > 0);// цикл по q M5

p /= 2;

iteration++;

} while (p > 0);// цикл по p M6

}

}

}