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 № 6**

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 узагальненого стеку/ 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).

Теоретичні відомості

Паралельне сортування Бетчера. Щоб отримати алгоритм обмінного сортування, час роботи якого має порядок, менший за N^2, необхідно підібрати для порівнянь пари несусідніх ключів (Kj, Кі); інакше доведеться виконати стільки операцій обміну записів, скільки інверсій є в початковій перестановці. Середнє число інверсій дорівнює ¼(N^2-N). У 1964 році К. Е. Бетчер відкрив цікавий спосіб програмування послідовності порівнянь, призначеної для пошуку можливих обмінів. Його метод далеко не очевидний. Справді, обґрунтувати його справедливість досить складно, оскільки виконується відносно мало порівнянь.

Схема сортування Бетчера дещо нагадує сортування Шелла, але порівняння виконуються по-новому, а тому ланцюжка операцій обміну записів не виникає. Сортування Бетчера діє, як 8-, 4-, 2- і 1-сортування, але порівняння не перекриваються. Оскільки в алгоритмі, по суті, відбувається злиття пар розсортованих підпослідовностей, його можна назвати обмінним сортуванням зі злиттям.

Вербальний опис алгоритму сортування М (Обмінне сортування зі злиттям)

Записи R1, ... Rn перекомпоновуються в межах того самого простору в пам'яті. Після завершення сортування їхні ключі будуть упорядковані: К1 ≤...≤ КN. Припускається, що N ≥ 2

**M1.** [Початкове встановлення р.] Встановити р 🡨 2t-1, де *t = lgN* - найменше ціле число, таке, що 2t ≥ N. (Кроки М2-М5 виконуватимуться з *р* = 2t-1, *р* = 2t-2, ... , ... , 1.)

**М2.** [Початкове встановлення *q, r, d*.] Встановити q 🡨2t-1, r 🡨 0, d 🡨 р.

**М3.** [Цикл по *i*.] Для всіх *i*, таких, що 0 ≤ *i* ≤ *N - d* і *i* /\ *р* = *r*, виконати крок М4. Потім перейти до кроку М5. (Тут через *i* /\ *р* позначено операцію "порозрядне логічне І" над поданнями цілих чисел *i* і *p*; усі біти результату дорівнюють 0, окрім тих бітів, для яких у відповідних розрядах *i* і *p* знаходяться 1. Так, 13 & 21 = (1101)2 /\ (10101)2 = (00101)2 = (00101)2 = 5. До цього моменту *d* - непарне кратне р (тобто частка від ділення *d* на *р* непарна), а *p* - ступінь двійки, тож i /\ р нерівно *(i + d) /\ р*. Звідси випливає, що крок М4 можна виконувати за всіх потрібних значень i в будь-якому порядку або навіть одночасно).

**М4.** [Порівняння/обмін Ri+1 : Ri+d+1] Якщо Ki+1 > Ki+d+1, поміняти місцями записи Ri+1 і Ri+d+1.

**М5.** [Цикл по *q*.] Якщо *q != р*, встановити *d =q - р*, *q = q/2*, *r = р* і повернутися до кроку М3.

**М6.** [Цикл по *p*.] (До цього моменту перестановка К1, К2 ... КN буде *p*-впорядкована.) Установити *р = [p/2]*. Якщо *р* > 0, повернутися до кроку М2.

Опис алгоритму сортування М (Обмінне сортування зі злиттям) у вигляді блок схеми

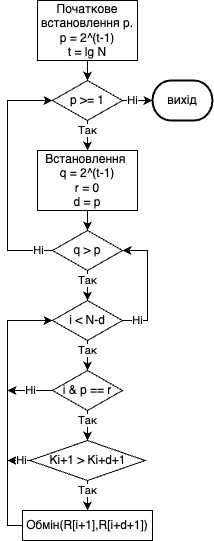


Рисунок 1 – Блок схема до алгоритму сортування

Приклад, для показу сутності алгоритму сортування чисел з докладним описом кожного кроку

1. Початкове встановлення:

N = 5, t = 3.

p = 2^(t-1) = 2^2 = 4.

1. Початкове встановлення q, r, d:

q = 4, r = 0, d = p = 4.

1. Цикл по i (M3):

* Ітерація 1 (i = 0):

Оскільки 0 /\ 4 = 0 = r, переходимо до кроку M4.

1. Порівняння/обмін (M4):

Порівнюємо K1 (5555) та K5 (1).

Оскільки K1 > K5, міняємо їх місцями, отримуючи [1, 4444, 333, 22, 5555].

1. Цикл по q (M5):

Збільшуємо r на q (r = 4), зменшуємо q удвічі (q = 2), зменшуємо d (d = 2), і повторюємо M3.

1. Цикл по i (повторення M3 та M4):

* Ітерація 2 (i = 0):

Оскільки 0 /\ 2 = 0 = r, переходимо до M4.

Порівнюємо K1 (1) та K3 (333).

Оскільки K1 < K3, залишаємо їх на місці.

* Ітерація 2 (i = 1):

Оскільки 1 /\ 2 = 0 = r, переходимо до M4.

Порівнюємо K2 (4444) та K4 (22).

Оскільки K2 > K4, міняємо їх місцями, отримуючи [1, 22, 333, 4444, 5555].

1. Цикл по q (повторення M5):

Збільшуємо r на q (r = 2), зменшуємо q удвічі (q = 1), зменшуємо d (d = 1), і повторюємо M3.

1. Цикл по i (повторення M3 та M4):

* Ітерація 3 (i = 0):

Оскільки 0 /\ 1 = 0 = r, переходимо до M4.

Порівнюємо K1 (1) та K2 (22).

Оскільки K1 < K2, залишаємо їх на місці.

* Ітерація 3 (i = 1):

Оскільки 1 /\ 1 = 1 ≠ r, не виконуємо M4.

* Ітерація 3 (i = 2):

Оскільки 2 /\ 1 = 0 = r, переходимо до M4.

Порівнюємо K3 (333) та K4 (4444).

Оскільки K3 < K4, залишаємо їх на місці.

1. Цикл по q (повторення M5):

Збільшуємо r на q (r = 1), зменшуємо q удвічі (q = 0), зменшуємо d (d = 0), і завершуємо M3.

Після виконання всіх ітерацій алгоритму M масив відсортований за зростанням [1, 22, 333, 4444, 5555].

Лістинг програми

using System.Text;

Console.OutputEncoding = Encoding.UTF8;

Console.InputEncoding = Encoding.UTF8;

Console.WriteLine("Для виходу введіть \"-0\"\n");

while (true)

{

try

{

int length = 0;

Console.Write("Введіть довжину масиву: ");

string input = Console.ReadLine();

if (input == "-0") Environment.Exit(0);

if (int.TryParse(input, out int temp))

if (input != null && temp >= 2)

length = temp;

else throw new Exception("Помилка введення кількості елементів масиву");

else throw new Exception("Помилка читання з рядка");

int[] array = GenerateRandomArray(length);

// Виведення початкового масиву

PrintArr("Початковий масив: ", array);

// Сортування масиву

MSort(array);

// Виведення відсортованого масиву

PrintArr("Відсортований масив: ", array);

Console.ReadLine();

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

}

static int[] GenerateRandomArray(int length)

{

Random random = new Random();

int[] array = new int[length];

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

array[i] = random.Next(-1000, 1000); // Генеруємо випадкові числа в діапазоні (-1000, 1000)

return array;

}

void PrintArr(string message, int[] array)

{

Console.WriteLine(message);

foreach (int el in array)

Console.Write($"{el} ");

Console.WriteLine();

}

void MSort(int[] 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

Console.WriteLine($"\nПочаткове встановлення M1: p = {p}");

do

{

Console.WriteLine($"\nІтерація: {iteration}");

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

int r = 0;

int d = p;

do

{

Console.WriteLine($"M2: q = {q}, r = {r}, d = {d}");

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

{

Console.WriteLine($"M3: i = {i}");

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

{

Console.WriteLine($"M4: Обмін {array[i]} <-> {array[i + d]}");

int temp\_swap = array[i];

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

array[i + d] = temp\_swap;

PrintColorArr("Результат перестановки:", array, i, i + d);

}

else Console.WriteLine($"M4: Обмін не потрібний {array[i]} < {array[i + d]}");

}

if (q != p)

{

d = q - p;

q /= 2;

r = p;

}

else

{

Console.WriteLine($"M5: Вихід з циклу, оскільки q == p ({q}={p})");

break; // вихід з циклу, якщо q=p

}

} while (q > 0);// цикл по q M5

p /= 2;

Console.WriteLine($"M6: p = {p}");

iteration++;

} while (p > 0);// цикл по p M6

}

void PrintColorArr(string message, int[] array, int highlightIndex1 = -1, int highlightIndex2 = -1)

{

Console.WriteLine(message);

for (int i = 0; i < array.Length; i++)

{

if (i == highlightIndex1 || i == highlightIndex2)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.Write($"{array[i]} ");

Console.ResetColor();

}

else

{

Console.Write($"{array[i]} ");

}

}

Console.WriteLine();

}

Екранні форми результатів роботи програми

Введення некоректної кількості елементів масиву. Результат подано на рисунку 2.

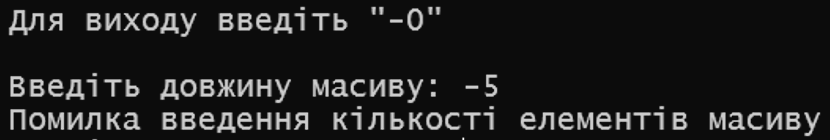


Рисунок 2 – перевірка на введення кількості елементів

Введення некоректних числових даних для кількості елементів масиву. (буквенні символи і т.п.). Результат на рисунку 3.

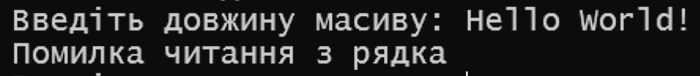


Рисунок 3 – поведінка при введенні не числових даних

Перевірка введення мінімального розміру масиву та його сортування. Результат на рисунку 4.

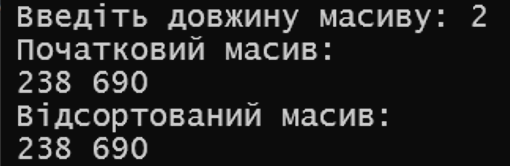


Рисунок 4 – робота з мінімальним масивом

Сортування масиву з 20 випадково згенерованими елементами. Результат на рисунку 5.

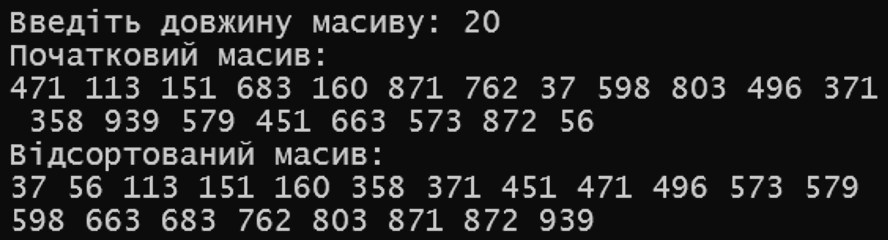


Рисунок 5 – масив з 20 елементами

Висновок

У ході виконання лабораторної роботи №6 було вивчено алгоритми сортування масивів чисел. Алгоритм було записано двома способами: у вербальній формі та у вигляді блок-схеми.

Суть алгоритму полягає в обміні сусідніх елементів масиву за умови, що перший елемент менший за другий. Цей процес повторюється для кожної пари сусідніх елементів масиву. Після кожного проходження масиву величина останнього елемента вже впорядкована, і такий процес повторюється до тих пір, поки не буде досягнуто повного упорядкування всього масиву.

При реалізації програми було використано генератор випадкових чисел для ініціалізації масиву даних. Окремий метод з параметрами був створений для реалізації алгоритму сортування.

У ході вивчення алгоритму та його реалізації була створена презентація, яка відображає ключові етапи та особливості алгоритму сортування масиву чисел.

Завдання дозволило краще зрозуміти принципи та механізми сортування масивів чисел, а також вивчити практичні аспекти реалізації цього алгоритму в програмному коді.