Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ(ІСТ)

Звіт

з лабораторної роботи № 3 з дисципліни
«Теорія алгоритмів»

„ Метод швидкого сортування ”

Виконали **ІА-31 Дук М.Д, Клим'юк В.Л, Сакун Д.С, Самелюк А.С**

Перевірив **Степанов А.С**

Київ 2024

Завдання

Реалізувати наступні три модифікації алгоритму швидкого сортування (Quick Sort) та порівняти їх швидкодію. Швидкість алгоритмів порівнюється на основі підрахунку кількості порівнянь елементів масиву під час роботи алгоритмів.

Алгоритм 1. Звичайний алгоритм швидкого сортування

В якості опорного елементу масиву під час кожного розбиття використовується останній елемент з поточного підмасиву (див. лекцію).

```
Partition(A, p, r)
1  x ← A[r]
2  i ← p - 1
3  for j ← p to r-1
4     do if A[j] ≤ x
5           then i ← i + 1
6                Обміняти A[i] ↔ A[j]
7  Обміняти A[i+1] ↔ A[r]
8  return i + 1
```

В цьому алгоритмі є тільки одне місце, де відбувається порівняння елементів масиву — рядок 4 наведеного вище псевдокоду. Зверніть увагу, що при додаванні в код лічильника порівнянь, місце, де повинно відбуватись його збільшення (інкрементація), повинно розташовуватись перед умовою if, а не в середині тіла умовного оператора (тобто після then).

Алгоритм 2. Швидке сортування з 3-медіаною в якості опорного елемента

Ця модифікація швидкого сортування працює наступним чином. Перед початком кожного розбиття для поточного підмасиву (A[p..r]) обирається три елементи: перший елемент підмасиву, останній елемент підмасиву, та елемент за індексом (p+r)/2 (той що знаходиться посередині підмасиву). Серед цих обраних елементів в якості опорного для подальшого розбиття обирається медіана — середній з трьох обраних.

При підрахунку кількості порівнянь даного алгоритму необхідно враховувати наступне.

• Порівняння не підраховуються під час визначення медіани.

• Процедура розбиття викликається тільки для підмасивів розміром більше 3. Для підмасивів з розміром менше або рівним 3, відбувається сортування без процедури розбиття. Але в цьому випадку необхідно всеодно враховувати порівняння елементів і вести їм облік.

На додаткові 2 бали:

Алгоритм 3. Швидке сортування з трьома опорними елементами

В цій модифікації замість одного опорного елементу обирається три. Позначимо ці опорні елементи q_1, q_2, q_3 (необхідно, щоб виконувалось: $q_1 < q_2 < q_3$). Перед основною частиною процедури розбиття ці опорні елементи обираються серед наступних елементів підмасиву A[p..r]: A[p], A[p+1] та A[r]. По завершенню розбиття всі елементи підмасиву $A[p..q_1-1]$ будуть менші за q_1, всі елементи $A[q_1+1..q_2-1]$ — менші за q_2, всі елементи $A[q_2+1..q_3-1]$ — менші за q_3,та всі елементи $A[q_3+1..r]$ — більші за q_3. І алгоритм рекурсивно продовжує свою роботу для вказаних чотирьох частин масиву: $A[p..q_1-1]$, $A[q_1+1..q_2-1]$, $A[q_2+1..q_3-1]$, $A[q_3+1..r]$.

```
namespace Classes
{
    public class Algorithm
    {
        private long _comparisons = 0L;

        //sort with last pivot
        public long QuickSort1(int[] arr)
        {
            _comparisons = 0L;
            _QuickSort1(arr, 0, arr.Length - 1);
            return _comparisons;
        }

        private void _QuickSort1(int[] arr, int l, int r)
        {
            if (l >= r) return;

            int pivot = _QuickSort1Split(arr, l, r);

            _QuickSort1(arr, l, pivot - 1);
            _QuickSort1(arr, pivot + 1, r);
        }

        private int _QuickSort1Split(int[] arr, int l, int r)
        {
            int pivot = arr[r];
            int index = l - 1;

            for (int i = l; i < r; i++)
            {
                this._comparisons++;
                if (arr[i] < pivot)
                {
                    index++;
                    (arr[index], arr[i]) = (arr[i], arr[index]);
                }
            }

            (arr[r], arr[index + 1]) = (arr[index + 1], arr[r]);
            return index + 1;
        }

        //sort with median of 3
        public long QuickSort2(int[] arr)
        {
            _comparisons = 0L;
            _QuickSort2(arr, 0, arr.Length - 1);
            return _comparisons;
        }

        private void _QuickSort2(int[] arr, int l, int r)
        {
```

```
            if (l >= r) return;

            if (r - l == 2)
            {
                int m = (l + r) / 2;

                SortThree(arr, l, m, r);

                return;
            }

            if (r - l == 1)
            {
                _comparisons++;
                if (arr[l] > arr[r])
                    (arr[l], arr[r]) = (arr[r], arr[l]);
                return;
            }

            int pivot = _QuickSort2Split(arr, l, r);

            _QuickSort2(arr, l, pivot - 1);
            _QuickSort2(arr, pivot + 1, r);
        }

        private int _QuickSort2Split(int[] arr, int l, int r)
        {
            int m = (l + r) / 2;

            SortThree(arr, l, m, r);

            int index = l + 1;
            int pivot = arr[m];
            int pivotIndex = r - 1;
            (arr[m], arr[pivotIndex]) = (arr[pivotIndex], arr[m]);

            for (int i = index; i < pivotIndex; i++)
            {
                _comparisons++;
                if (arr[i] < pivot)
                {
                    (arr[index], arr[i]) = (arr[i], arr[index]);
                    index++;
                }
            }

            (arr[index], arr[pivotIndex]) = (arr[pivotIndex], arr[index]);

            return index;
        }

        //Sort three items at given indices
        private void SortThree(int[] arr, int l, int m, int r)
```

```
        {
            _comparisons += 3;
            int min;
            if (arr[m] < arr[r])
            {
                min = m;
            }
            else
            {
                min = r;
            }

            if (arr[l] > arr[min])
            {
                (arr[l], arr[min]) = (arr[min], arr[l]);
            }

            if (arr[m] > arr[r])
            {
                (arr[m], arr[r]) = (arr[r], arr[m]);
            }
        }

        //sort with 3 pivot elements
        public long QuickSort3(int[] arr)
        {
            _comparisons = 0L;
            _QuickSort3(arr, 0, arr.Length - 1);
            return _comparisons;
        }
        private void _QuickSort3(int[] arr, int l, int r)
        {
            if (l >= r) return;

            if (r - l == 2)
            {
                int m = (l + r) / 2;

                SortThree(arr, l, m, r);

                return;
            }

            if (r - l == 1)
            {
                _comparisons++;
                if (arr[l] > arr[r])
                    (arr[l], arr[r]) = (arr[r], arr[l]);
                return;
            }

            int pivot = _QuickSort3Split(arr, l, r);
```

```
            _QuickSort3(arr, l, pivot - 1);
            _QuickSort3(arr, pivot + 1, r);
        }

        private int _QuickSort3Split(int[] arr, int l, int r)
        {
            int a = l + 2, b = l + 2, c = r - 1, d = r - 1;
            int p = arr[l], q = arr[l + 1], pivot = arr[r];

            while (b <= c)
            {
                _comparisons++;
                for(; arr[b] < q && b <= c; _comparisons++)
                {
                    _comparisons++;
                    if (arr[b] < p)
                    {
                        (arr[a], arr[b]) = (arr[b], arr[a]);
                        a++;
                    }
                    b++;
                }

                _comparisons++;

                for(; arr[c] > q && b <= c; _comparisons++)
                {
                    _comparisons++;
                    if (arr[c] > pivot)
                    {
                        (arr[c], arr[d]) = (arr[d], arr[c]);
                        d--;
                    }
                    c--;
                }

                if (b <= c)
                {
                    _comparisons++;
                    if (arr[b] > pivot)
                    {
                        _comparisons++;
                        if (arr[c] < p)
                        {
                            (arr[b], arr[a]) = (arr[a], arr[b]);
                            (arr[a], arr[c]) = (arr[c], arr[a]);
                            a++;
                        }
                        else
                        {
                            (arr[b], arr[c]) = (arr[c], arr[b]);
                        }
                        (arr[c], arr[d]) = (arr[d], arr[c]);
```

```
                    b++;
                    c--;
                    d--;
                }
                else
                {
                    _comparisons++;
                    if (arr[c] < p)
                    {
                        (arr[b], arr[a]) = (arr[a], arr[b]);
                        (arr[a], arr[c]) = (arr[c], arr[a]);
                        a++;
                    }
                    else
                    {
                        (arr[b], arr[c]) = (arr[c], arr[b]);
                    }
                    b++;
                    c--;
                }
            }
        }

        a--;
        b--;
        d++;
        (arr[l + 1], arr[a]) = (arr[a], arr[l + 1]);
        (arr[a], arr[b]) = (arr[b], arr[a]);
        a--;
        (arr[l], arr[a]) = (arr[a], arr[l]);
        (arr[r], arr[d]) = (arr[d], arr[r]);

        return a;
    }

}
}
```

Код 1.1 – Реалізація алгоритмів швидкого сортування

```
using Classes;

namespace App
{
    internal class Program
    {
        static async Task Main(string[] args)
        {
            string[] files = new string[]
            {
                "input_01_10.txt",
                "input_02_10.txt",
                "input_03_10.txt",
```

```
                "input_04_10.txt",
                "input_05_10.txt",
                "input_06_100.txt",
                "input_07_100.txt",
                "input_08_100.txt",
                "input_09_100.txt",
                "input_10_100.txt",
                "input_11_1000.txt",
                "input_12_1000.txt",
                "input_13_1000.txt",
                "input_14_1000.txt",
                "input_15_1000.txt",
                "input_16_10000.txt",
                "input_17_10000.txt",
                "input_18_10000.txt",
                "input_19_10000.txt",
                "input_20_10000.txt"
            };

            FileHelper helper = new FileHelper();
            Algorithm alg = new Algorithm();
            foreach (var file in files)
            {
                string root = "D:\\GITHUB\\teoriya-algoritmiv-3\\Classes\\tests";
                int[] newarr1 = await helper.ReadFile(Path.Combine(root, "input",
file));
                int[] newarr2 = (int[])newarr1.Clone();
                int[] newarr3 = (int[])newarr1.Clone();

                long c1 = alg.QuickSort1(newarr1);
                long c2 = alg.QuickSort2(newarr2);
                long c3 = alg.QuickSort3(newarr3);

                string file_out = file.Replace("input", "output");

                await helper.WriteFile(Path.Combine(root, "output", file_out), c1,
c2, c3);
            }

            //Test random

            //int[] arr = CreateArray(10000);

            //Algorithm a = new Algorithm();
            //long comps = a.QuickSort3(arr);

            //Console.WriteLine("Comparisons: " + comps);
            //foreach (var i in arr)
            //{
            //    Console.WriteLine(i);
            //}
        }
```

```
        static int[] CreateArray(long size)
        {
            int[] res = new int[size];

            for (long i = 0; i < size; i++)
            {
                res[i] = Random.Shared.Next(0, 10000);
            }

            return res;
        }
    }
}
```

Код 1.2 – Опрацювання вхідних файлів

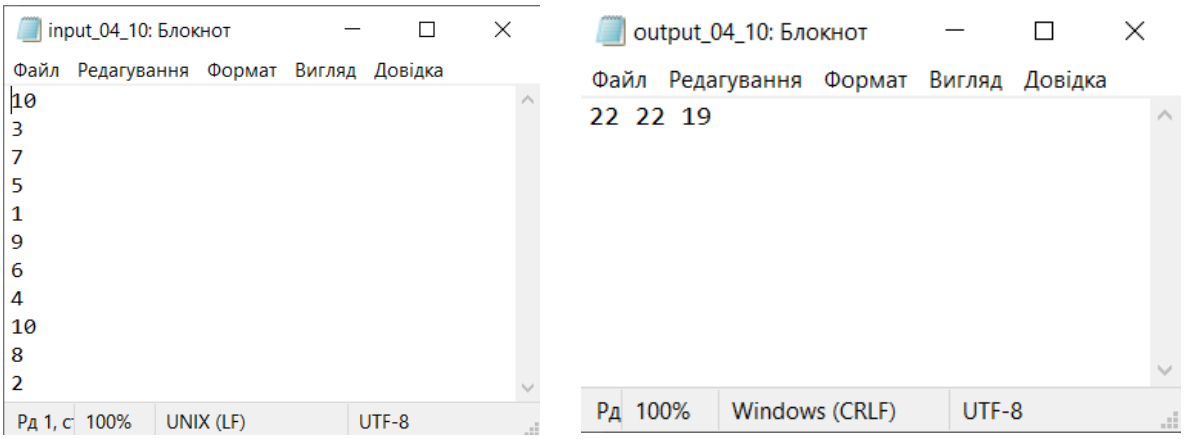

Рисунок 1.(1-2) - Приклад вхідних і вихідних даних

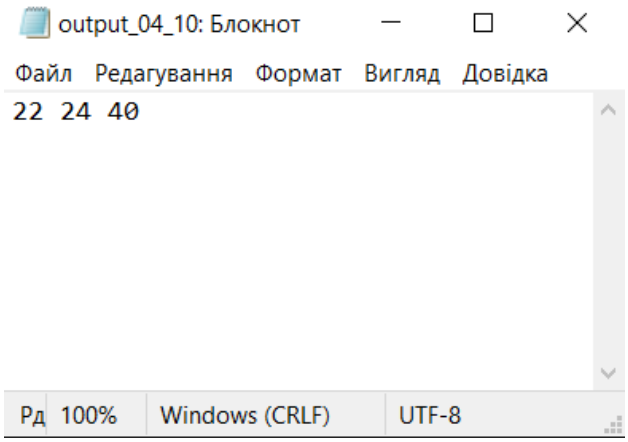

Рисунок 1.3 - Приклад роботи коду

Висновок: В ході лабораторної роботи було проведено аналіз та порівняння швидкого сортування з 3-медіаною в якості опорного елемента та швидкого сортування з трьома опорними елементами. Результати вказують на те, що використання 3-медіани як опорного елемента забезпечує більш стабільну та ефективну роботу алгоритму в порівнянні з використанням трьох опорних елементів. Цей підхід дозволяє зменшити кількість порівнянь та обмінів, що пришвидшує процес сортування. Таким чином, використання 3-медіани в якості опорного елемента виявляється більш оптимальним у контексті швидкого сортування. А також кількість порівнянь в вихідних файлах даних для перевірки може відрізнятися від кількості порівнянь, порахованих нашим алгоритмом, так як метод №2 та №3 можуть бути реалізовані різним чином.