Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
ФІОТ
ІСТ

Лабораторна робота №10

по курсу *“*Абстрактні типи даних”

Варіант 3

Виконали ст. групи ІС-31:
Коваль Богдан
Михайлова Софія
Шмигельський Ілля
Перевірив: ст. вик. Дорошенко К.С.

Київ 2024

Комп’ютерний практикум

Тема: Алгоритми сортування. Методи сортування малих обсягів даних.

Мета роботи: Дослідження та порівняння алгоритмів сортування.

Завдання

Мережу зарядних пристроїв вирішили оновити. Але це треба робити поступово,

замінюючи пристрої в порядку збільшення їх потужності. Фахівці зробили перелік, але

з’ясувалося, що після цього додалася додаткова частина мережі, яку необхідно включити до

списку. Відбулося переформатування списку. Раптово компанія вирішила об’єднати всі

мережі. Виявилося, що в цій купі пристроїв є багато з однаковою потужністю. Зробили

остаточний перелік. Допоможіть сформувати перелік пристроїв для кожного з трьох випадків.

Склад звіту практичної роботи

– постановка задачі;

– опис швидкого алгоритму сортування;

– блок-схема реалізації, на якій виконано аналіз складності алгоритму;

– результати дослідження у вигляді графіків в координатах;

– висновки про доцільність використання кожного з алгоритмів для даних задачі

та про відповідність результатів експериментального дослідження аналітичним

оцінкам складності;

– відповіді на контрольні питання.

Варіанти завдань.

Мережу зарядних пристроїв вирішили оновити. Але це треба робити поступово,

замінюючи пристрої в порядку збільшення їх потужності. Фахівці зробили перелік, але

з'ясувалося, що після цього додалася додаткова частина мережі, яку необхідно включити до

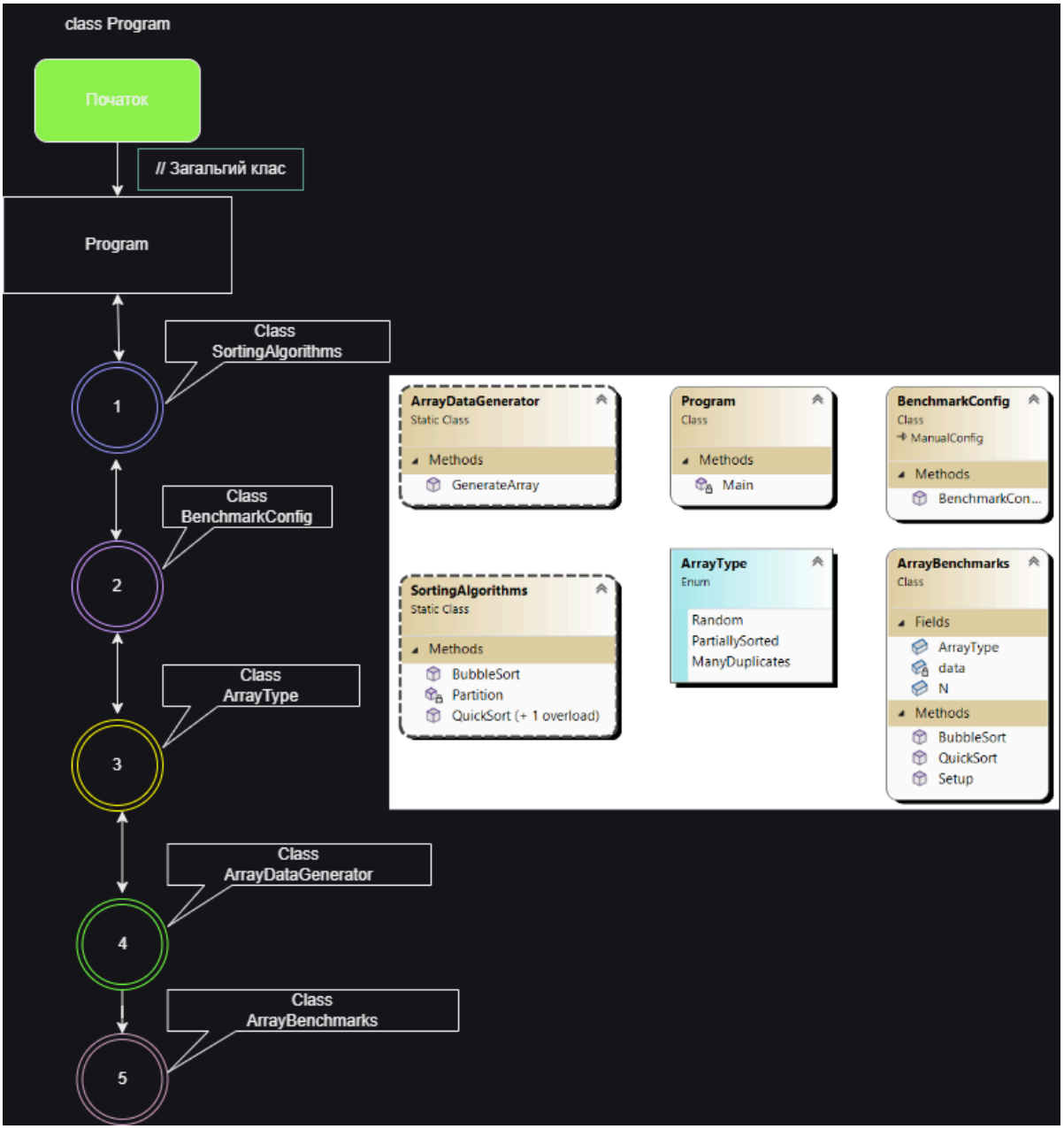
списку. Відбулося переформатування списку. Раптово компанія вирішила об'єднати всі

мережі. Виявилося, що в цій купі пристроїв є багато з однаковою потужністю. Зробили

остаточний перелік. Допоможіть сформувати перелік пристроїв для кожного з трьох випадків,

обираючи простий метод сортування.

Побудова моделі

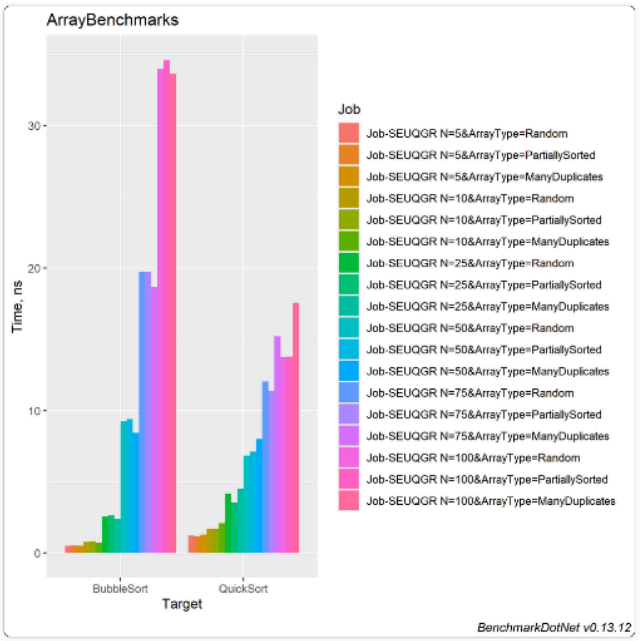


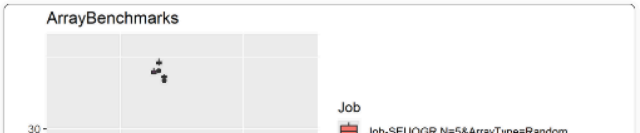
Інтерфейс користувача

```
BenchmarkDotNet v0.13.12, Windows 10 (10.0.19045.4412/22H2/2022Update)
12th Gen Intel Core i5-12450H, 1 CPU, 12 logical and 8 physical cores
.NET SDK 8.0.100 preview.3.24204.13
  [Host]     : .NET 5.0.17 (5.0.1722.21314), X64 RyuJIT AVX2
  Job-SEUQGR : .NET 5.0.17 (5.0.1722.21314), X64 RyuJIT AVX2

IterationCount=10  RunStrategy=Throughput  WarmupCount=5
```

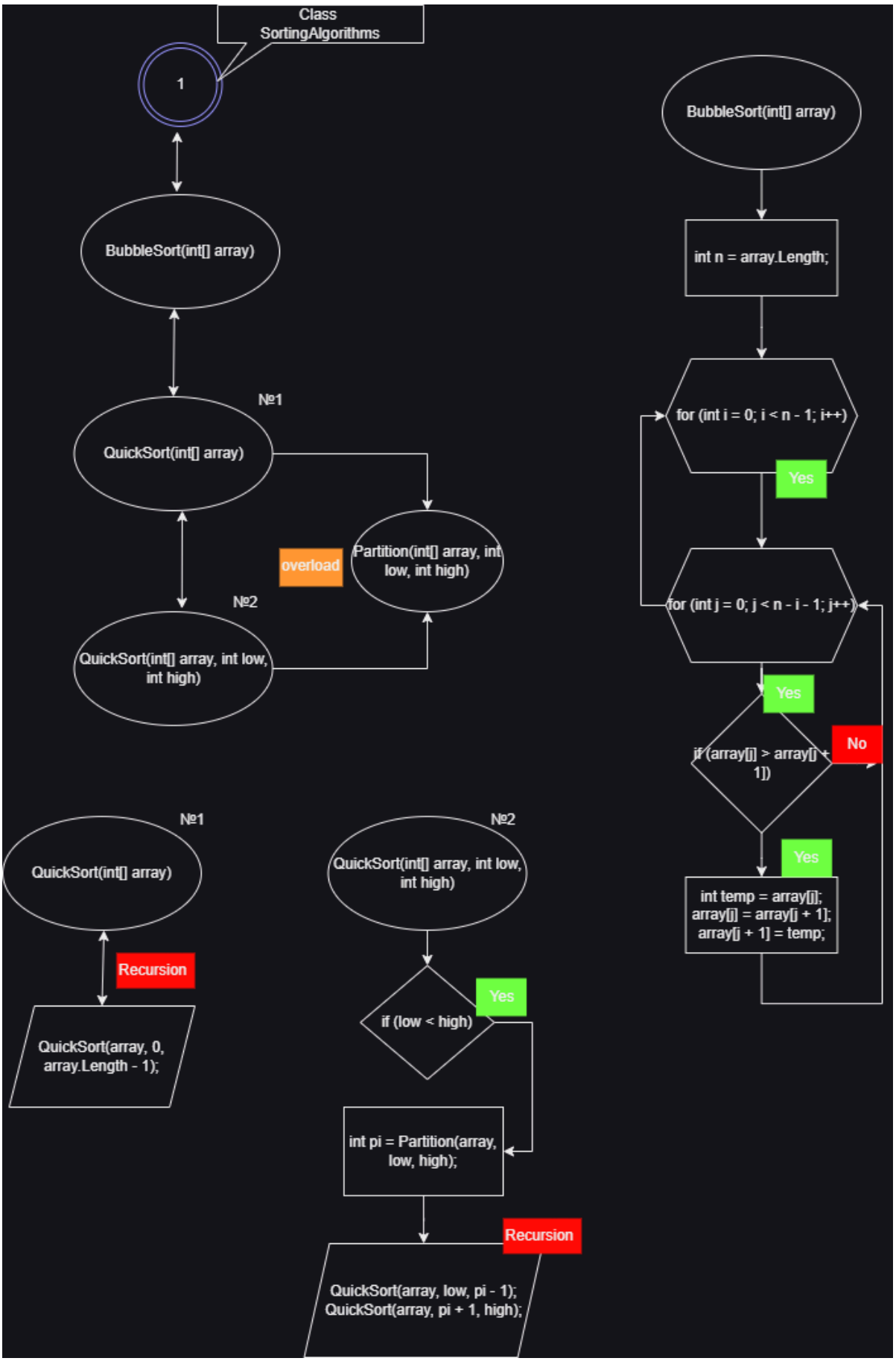
Method	N	ArrayType	Mean	Error	StdDev	Gen0	Allocated
BubbleSort	5	Random	0.5224 ns	0.0049 ns	0.0033 ns	0.0001	-
QuickSort	5	Random	1.2220 ns	0.0264 ns	0.0174 ns	0.0002	1 B
BubbleSort	5	PartiallySorted	0.5336 ns	0.0198 ns	0.0131 ns	0.0001	-
QuickSort	5	PartiallySorted	1.1897 ns	0.0129 ns	0.0077 ns	0.0002	1 B
BubbleSort	5	ManyDuplicates	0.5099 ns	0.0036 ns	0.0024 ns	0.0001	-
QuickSort	5	ManyDuplicates	1.2988 ns	0.0086 ns	0.0057 ns	0.0002	1 B
BubbleSort	10	Random	0.7900 ns	0.0032 ns	0.0017 ns	0.0001	1 B
QuickSort	10	Random	1.7164 ns	0.0090 ns	0.0053 ns	0.0002	1 B
BubbleSort	10	PartiallySorted	0.8203 ns	0.0152 ns	0.0100 ns	0.0001	1 B
QuickSort	10	PartiallySorted	1.7211 ns	0.0219 ns	0.0130 ns	0.0002	1 B
BubbleSort	10	ManyDuplicates	0.7176 ns	0.0067 ns	0.0044 ns	0.0001	1 B
QuickSort	10	ManyDuplicates	2.0890 ns	0.0145 ns	0.0096 ns	0.0002	1 B
BubbleSort	25	Random	2.5571 ns	0.0725 ns	0.0479 ns	0.0002	1 B
QuickSort	25	Random	4.1689 ns	0.1219 ns	0.0806 ns	0.0004	3 B
BubbleSort	25	PartiallySorted	2.6486 ns	0.0411 ns	0.0245 ns	0.0002	1 B
QuickSort	25	PartiallySorted	3.5381 ns	0.0568 ns	0.0376 ns	0.0004	3 B
BubbleSort	25	ManyDuplicates	2.3959 ns	0.0825 ns	0.0546 ns	0.0002	1 B
QuickSort	25	ManyDuplicates	4.5115 ns	0.0709 ns	0.0422 ns	0.0004	3 B
BubbleSort	50	Random	9.2632 ns	0.1534 ns	0.0913 ns	0.0004	2 B
QuickSort	50	Random	6.8222 ns	0.0849 ns	0.0561 ns	0.0007	4 B
BubbleSort	50	PartiallySorted	9.3947 ns	0.1366 ns	0.0903 ns	0.0004	2 B
QuickSort	50	PartiallySorted	7.1274 ns	0.3129 ns	0.1862 ns	0.0007	4 B
BubbleSort	50	ManyDuplicates	8.4246 ns	0.0706 ns	0.0420 ns	0.0004	2 B
QuickSort	50	ManyDuplicates	8.0009 ns	0.0528 ns	0.0314 ns	0.0007	4 B
BubbleSort	75	Random	19.7316 ns	0.1313 ns	0.0781 ns	0.0005	3 B
QuickSort	75	Random	12.0394 ns	0.5480 ns	0.3625 ns	0.0010	7 B
BubbleSort	75	PartiallySorted	19.7471 ns	0.4060 ns	0.2123 ns	0.0005	3 B
QuickSort	75	PartiallySorted	11.3733 ns	0.1008 ns	0.0600 ns	0.0010	7 B
BubbleSort	75	ManyDuplicates	18.6649 ns	0.3219 ns	0.1915 ns	0.0005	3 B
QuickSort	75	ManyDuplicates	15.2075 ns	0.5589 ns	0.3697 ns	0.0010	7 B
BubbleSort	100	Random	33.9526 ns	0.2140 ns	0.1415 ns	0.0006	4 B
QuickSort	100	Random	13.7528 ns	0.1456 ns	0.0867 ns	0.0013	8 B
BubbleSort	100	PartiallySorted	34.5692 ns	0.4130 ns	0.2458 ns	0.0006	4 B
QuickSort	100	PartiallySorted	13.7932 ns	0.1607 ns	0.0956 ns	0.0013	8 B
BubbleSort	100	ManyDuplicates	33.6194 ns	0.2179 ns	0.1297 ns	0.0006	4 B
QuickSort	100	ManyDuplicates	17.5483 ns	0.5038 ns	0.2998 ns	0.0013	8 B

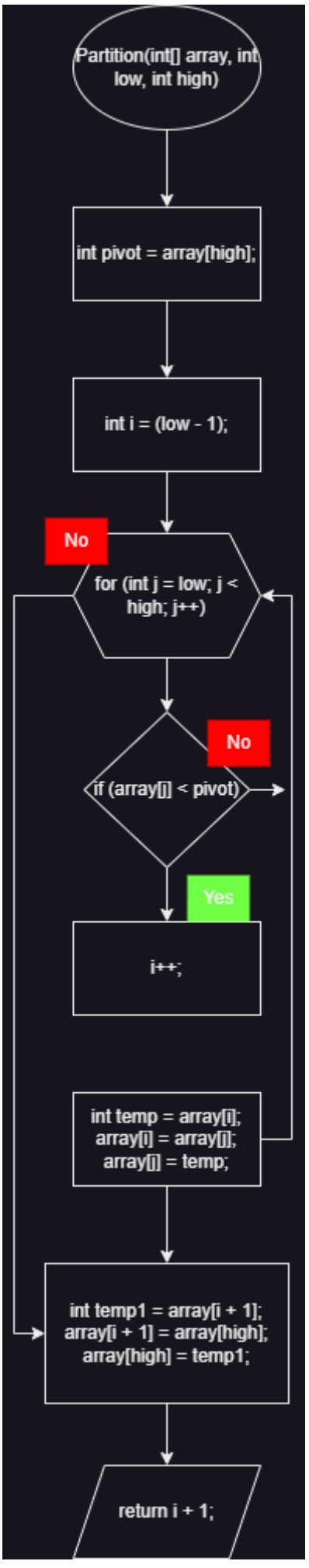




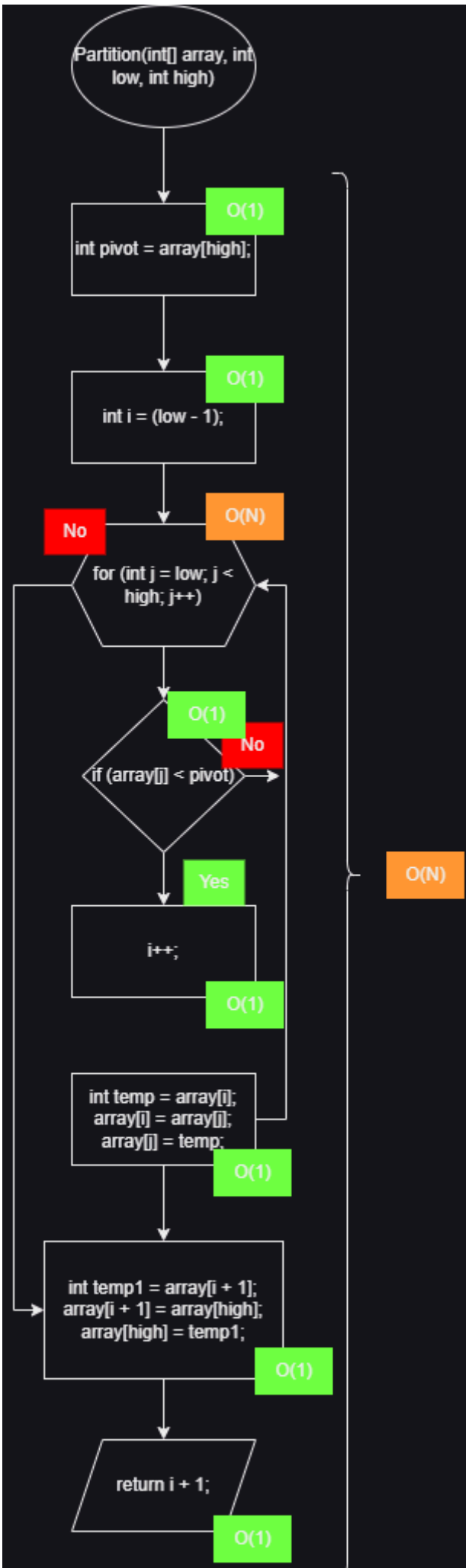
Інтерфейс виявлення помилок

Розробка алгоритму

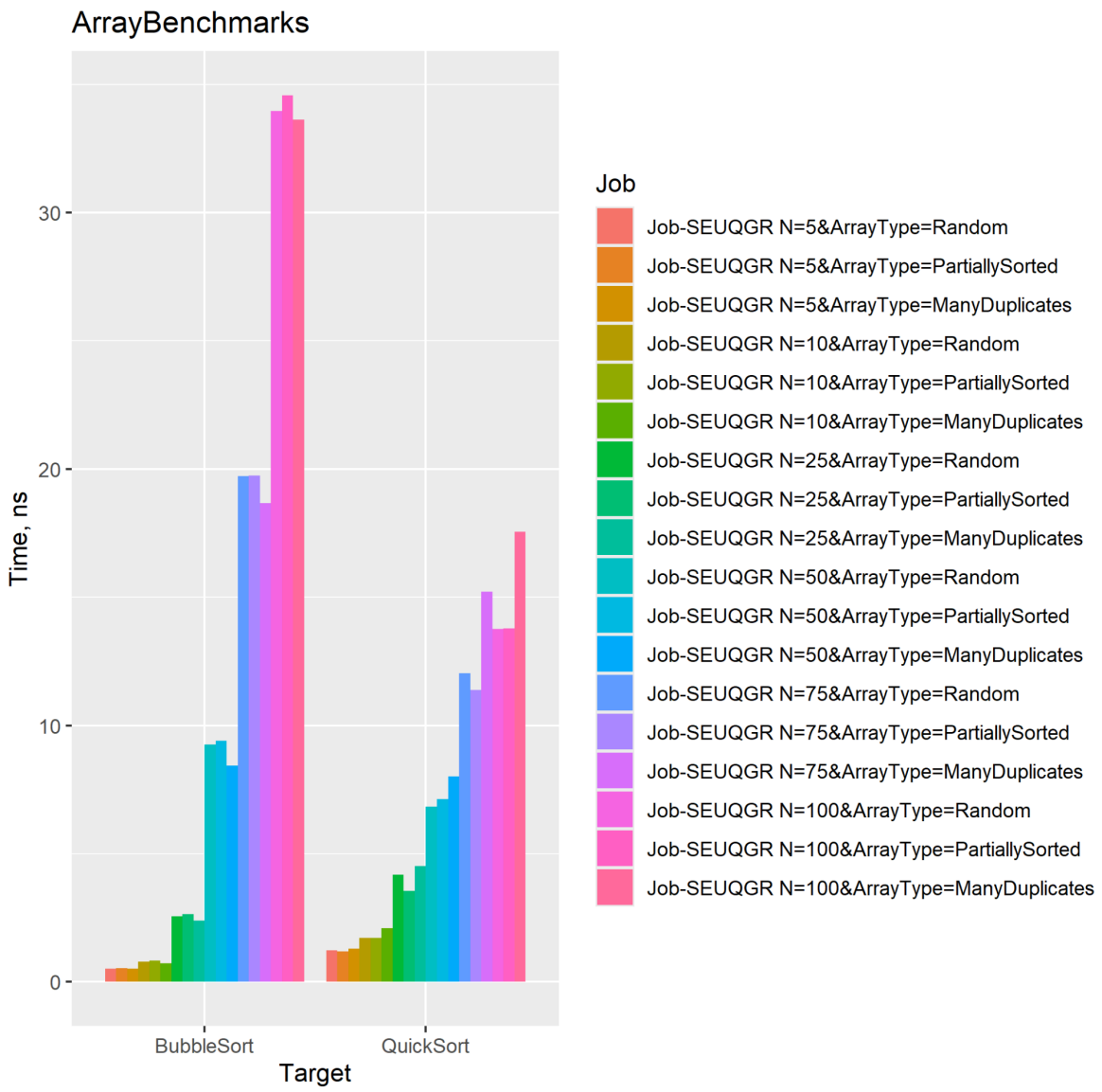

Class SortingAlgorithms
1
BubbleSort(int[] array)
№1
QuickSort(int[] array)
overload
Partition(int[] array, int low, int high)
№2
QuickSort(int[] array, int low, int high)
BubbleSort(int[] array)
int n = array.Length;
for (int i = 0; i < n - 1; i++)
Yes
for (int j = 0; j < n - i - 1; j++)
Yes
if (array[j] > array[j + 1])
No
Yes
int temp = array[j];
array[j] = array[j + 1];
array[j + 1] = temp;
№1
QuickSort(int[] array)
Recursion
QuickSort(array, 0, array.Length - 1);
№2
QuickSort(int[] array, int low, int high)
if (low < high)
Yes
int pi = Partition(array, low, high);
Recursion
QuickSort(array, low, pi - 1);
QuickSort(array, pi + 1, high);


Partition(int[] array, int low, int high)
int pivot = array[high];
int i = (low - 1);
No
for (int j = low; j < high; j++)
No
if (array[j] < pivot)
Yes
i++;
int temp = array[i];
array[i] = array[j];
array[j] = temp;
int temp1 = array[i + 1];
array[i + 1] = array[high];
array[high] = temp1;
return i + 1;

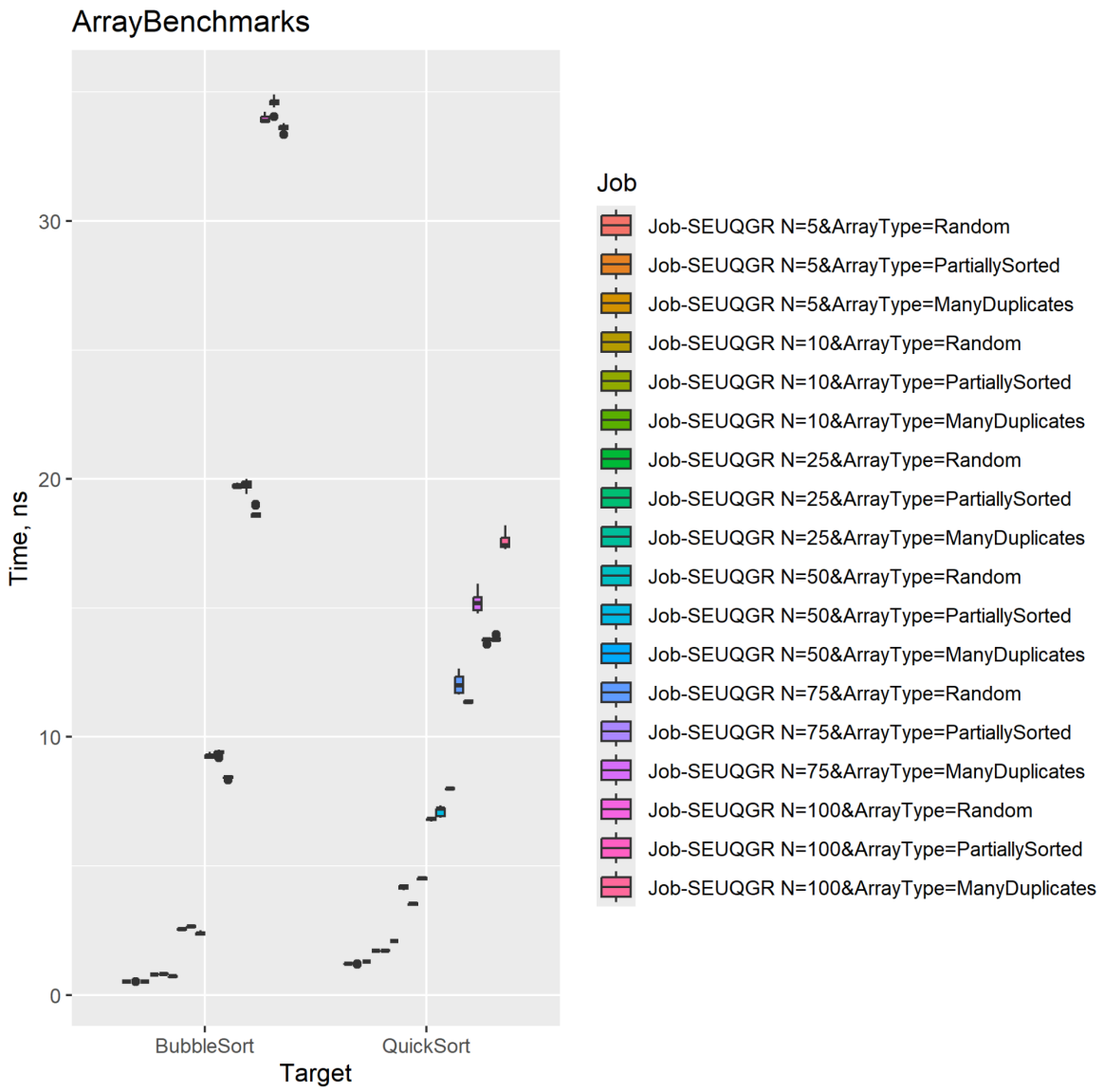
Аналіз алгоритму та його складності


Partition(int[] array, int low, int high)
O(1)
int pivot = array[high];
O(1)
int i = (low - 1);
O(N)
No
for (int j = low; j < high; j++)
O(1)
No
if (array[j] < pivot)
Yes
i++;
O(1)
O(N)
int temp = array[i];
array[i] = array[j];
array[j] = temp;
O(1)
int temp1 = array[i + 1];
array[i + 1] = array[high];
array[high] = temp1;
O(1)
return i + 1;
O(1)

Загальний графік оцінки складності



BenchmarkDotNet v0.13.12



BenchmarkDotNet v0.13.12

Порівняльний висновок :

У ході даної лабораторної роботи ми дослідили та порівняли продуктивність двох алгоритмів сортування – бульбашкового сортування та швидкого сортування – для різних типів даних. Мета роботи полягала у визначенні ефективності цих алгоритмів при роботі з хаотичними даними, частково впорядкованими даними та даними з багатьма повторами.

Основні висновки:

1. **Порівняння продуктивності:**

- **Хаотичні дані:** Швидке сортування виявилося значно продуктивнішим у порівнянні з бульбашковим сортуванням, демонструючи значно менший час виконання завдяки своїй O(n log n) складності у середньому випадку.
- **Частково впорядковані дані:** Швидке сортування також показало кращі результати, однак різниця з бульбашковим сортуванням була менш вираженою. Бульбашкове сортування може виявитися досить ефективним у випадках, коли масив вже майже відсортований, завдяки можливості раннього завершення.
- **Дані з багатьма повторами:** Швидке сортування знову продемонструвало кращу продуктивність. В той час як бульбашкове сортування затримувалось через численні порівняння та обміни, швидке сортування ефективніше обробляло повторювані елементи.

2. **Оцінка складності алгоритмів:**
   - **Бульбашкове сортування:** Має час виконання O(n^2) у найгіршому випадку. Це робить його непридатним для великих масивів даних, проте може бути корисним для невеликих або майже відсортованих масивів.
   - **Швидке сортування:** Має середній час виконання O(n log n), що робить його значно більш ефективним для великих масивів даних. Однак у найгіршому випадку (наприклад, для вже відсортованого масиву без оптимізації вибору опорного елемента) його складність також може досягати O(n^2).
3. **Графічний аналіз:**
   - Побудовані графіки для трьох типів даних (хаотичні, частково впорядковані, з багатьма повторами) продемонстрували чіткі переваги швидкого сортування у більшості випадків. Криві часу виконання для швидкого сортування були нижчими у порівнянні з кривими для бульбашкового сортування.

Загальний висновок:

Швидке сортування значно перевершує бульбашкове сортування за продуктивністю для всіх розглянутих типів даних, особливо для великих і хаотичних масивів. Бульбашкове сортування може бути доцільним лише для дуже малих обсягів даних або майже відсортованих масивів, де його проста реалізація може бути перевагою. Вибір алгоритму

сортування повинен враховувати характеристики даних та вимоги до продуктивності, де швидке сортування є універсальним і ефективним вибором.

Реалізація алгоритму:

```
using System;


namespace Benchmark
{
    public static class SortingAlgorithms
    {
    public static void BubbleSort(int[] array)
    {
        int n = array.Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (array[j] > array[j + 1])
                {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    public static void QuickSort(int[] array)
    {
        QuickSort(array, 0, array.Length - 1);
    }
```

```
    private static void QuickSort(int[] array, int low, int high)
    {
        if (low < high)
        {
            int pi = Partition(array, low, high);

            QuickSort(array, low, pi - 1);
            QuickSort(array, pi + 1, high);
        }
    }

    private static int Partition(int[] array, int low, int high)
    {
        int pivot = array[high];
        int i = (low - 1);

        for (int j = low; j < high; j++)
        {
            if (array[j] < pivot)
            {
                i++;

                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }

        int temp1 = array[i + 1];
        array[i + 1] = array[high];
        array[high] = temp1;

        return i + 1;
    }
```

```
    }

}
```

Контрольні запитання

1. Порівняльна характеристика методів сортування вибором, сортування вставками, бульбашкового сортування. На яких вхідних даних ці методи працюють найкраще?

- Сортування вибором (Selection Sort)
    - **Принцип**: Вибирає найменший або найбільший елемент з решти масиву та розміщує його у відсортовану частину.
    - **Часова складність**: $O(n^2)$.
    - **Просторова складність**: O(1), оскільки не вимагає додаткової пам'яті.
    - **Переваги**: Простий у реалізації та не змінює порядок рівних елементів (нестійкий).
    - **Недоліки**: Повільний для великих масивів, завжди виконує однакову кількість операцій.
    - **Найкраще працює**: Для невеликих масивів і даних з малою кількістю елементів.
- Сортування вставками (Insertion Sort)
    - **Принцип**: Вставляє кожен елемент у відсортовану частину масиву на його правильне місце.
    - **Часова складність**: $O(n^2)$ у гіршому випадку, O(n) у найкращому (масив майже відсортований).
    - **Просторова складність**: O(1).
    - **Переваги**: Стійкий, ефективний для невеликих масивів і майже відсортованих даних.
    - **Недоліки**: Повільний для великих масивів, особливо якщо дані відсортовані в зворотному порядку.
    - **Найкраще працює**: Для невеликих або майже відсортованих масивів.
- Бульбашкове сортування (Bubble Sort)

- **Принцип**: Послідовно порівнює та обмінює сусідні елементи, "спливаючи" найменші або найбільші до кінця масиву.
- **Часова складність**: $O(n^2)$, але може бути $O(n)$, якщо вже відсортований.
- **Просторова складність**: $O(1)$.
- **Переваги**: Стійкий, простий у реалізації, може бути ефективним, якщо масив майже відсортований.
- **Недоліки**: Неефективний для великих масивів, часто більше операцій, ніж необхідно.
- **Найкраще працює**: Для невеликих або майже відсортованих масивів.

2. Порівняльна характеристика методів сортування вибором і сортування методом підрахунку. На яких вхідних даних ці методи працюють найкраще?

- Сортування вибором (Selection Sort)
  - Дивіться попередній розділ.
  - **Найкраще працює**: Для невеликих масивів з довільними даними.
- Сортування методом підрахунку (Counting Sort)
  - **Принцип**: Підраховує кількість елементів кожного значення і використовує цю інформацію для сортування.
  - **Часова складність**: $O(n + k)$, де n — кількість елементів, k — діапазон значень.
  - **Просторова складність**: Залежить від діапазону значень, оскільки потрібно створити додатковий масив для підрахунку.
  - **Переваги**: Дуже швидкий для невеликих діапазонів, стійкий.
  - **Недоліки**: Вимагає додаткової пам'яті, обмежений діапазоном значень, не підходить для складних структур даних.
  - **Найкраще працює**: Для великих масивів з обмеженим діапазоном значень, як-от сортування оцінок або кодів.

3. Порівняння непрямого та прямого сортування вибором. На яких вхідних даних ці методи працюють найкраще?

- **Пряме сортування вибором (Selection Sort)**
    - Сортування виконується шляхом вибору найменшого/найбільшого елементу з решти масиву та розміщення його у відсортованій частині.
    - **Найкраще працює**: Для невеликих масивів.
- **Непряме сортування вибором (Indirect Selection Sort)**
    - Замість сортування оригінального масиву сортуються індекси або покажчики на елементи, зберігаючи оригінальний порядок у разі рівних елементів.
    - **Найкраще працює**: Для сортування великих масивів, особливо коли елементи містять багато даних, а їх порядок важливий.

4. Переваги та недоліки швидкого сортування для різних послідовностей даних. На яких вхідних даних цей метод працює найкраще?

- **Швидке сортування (QuickSort)**
    - **Принцип**: Розділяє масив на дві частини за допомогою "опорного" елемента (pivot), сортує ці частини рекурсивно, а потім об'єднує.
    - **Часова складність**: O(n log n) у середньому, $O(n^2)$ у найгіршому випадку.
    - **Просторова складність**: O(log n), якщо використовуються рекурсивні виклики.
    - **Переваги**: Швидкий для більшості даних, простий у реалізації, добре підходить для великих масивів.
    - **Недоліки**: Може бути повільним для вже відсортованих або майже відсортованих даних, особливо з поганим вибором "опорного" елементу.
    - **Найкраще працює**: Для випадкових або невідсортованих масивів. Для великих масивів з ефективним вибором "опорного" елементу.

Висновок: Лабораторна робота з теми "Алгоритми сортування. Прості методи сортування" була спрямована на вивчення та порівняння базових алгоритмів сортування, зокрема методів бульбашкового сортування, сортування вибором та

сортування вставками. Головна мета полягала в аналізі ефективності цих алгоритмів в різних умовах та на різних наборах даних.

Ця лабораторна робота допомогла глибше зрозуміти особливості простих алгоритмів сортування та їхній вплив на продуктивність. У результаті ми отримали базовий набір знань, який може бути використаний для подальшого вивчення та застосування більш складних методів сортування.