**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра ІПІ**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

„ **Проектування і аналіз алгоритмів внутрішнього сортування**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Головня Олександр Ростиславович*

**Перевірив**

(прізвище, ім'я, по батькові)

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc69772242)

[2 ЗаВдання 4](#_Toc69772243)

[3 Виконання 7](#_Toc69772244)

[3.1 Аналіз алгоритму на відповідність властивостям 7](#_Toc69772245)

[3.2 Псевдокод алгоритму 7](#_Toc69772246)

[3.3 Аналіз часової складності 7](#_Toc69772247)

[3.4 Програмна реалізація алгоритму 7](#_Toc69772248)

[3.4.1 Вихідний код 7](#_Toc69772249)

[3.4.2 Приклад роботи 8](#_Toc69772250)

[3.5 Тестування алгоритму 9](#_Toc69772251)

[3.5.1 Часові характеристики оцінювання 9](#_Toc69772252)

[3.5.2 Графіки залежності часових характеристик оцінювання від розмірності масиву 11](#_Toc69772253)

[Висновок 12](#_Toc69772254)

[Критерії оцінювання 13](#_Toc69772255)

# Мета лабораторної роботи

Мета роботи – вивчити основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінити поріг їх ефективності.

# ЗаВдання

Виконати аналіз алгоритму внутрішнього сортування на відповідність наступним властивостям (таблиця 2.1):

* стійкість;
* «природність» поведінки (Adaptability);
* базуються на порівняннях;
* необхідність додаткової пам'яті (об'єму);
* необхідність в знаннях про структуру даних.

Записати алгоритм внутрішнього сортування за допомогою псевдокоду (чи іншого способу по вибору).

Провести аналіз часової складності в гіршому, кращому і середньому випадках та записати часову складність в асимптотичних оцінках.

Виконати програмну реалізацію алгоритму на будь-якій мові програмування з фіксацією часових характеристик оцінювання (кількість порівнянь, кількість перестановок, глибина рекурсивного поглиблення та інше в залежності від алгоритму).

Провести ряд випробувань алгоритму на масивах різної розмірності (10, 100, 1000, 5000, 10000, 20000, 50000 елементів) і різних наборів вхідних даних (впорядкований масив, зворотно упорядкований масив, масив випадкових чисел) і побудувати графіки залежності часових характеристик оцінювання від розмірності масиву, нанести на графік асимптотичну оцінку гіршого і кращого випадків для порівняння.

Зробити порівняльний аналіз двох алгоритмів.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Алгоритм сортування** |
| 1 | Сортування бульбашкою |
| 2 | Сортування гребінцем («розчіскою») |

# Виконання

## Аналіз алгоритму на відповідність властивостям

Аналіз алгоритму сортування бульбашкою на відповідність властивостям наведено в таблиці 3.1.

Таблиця 3.1 – Аналіз алгоритму на відповідність властивостям

|  |  |  |
| --- | --- | --- |
| **Властивість** | **Сортування бульбашкою** | **Сортування гребінцем** |
| Стійкість | Так | Ні |
| «Природність» поведінки (Adaptability) | Ні | Ні |
| Базуються на порівняннях | так | так |
| Необхідність в додатковій пам'яті (об'єм) | 1 | 1 |
| Необхідність в знаннях про структури даних | так | так |

## Псевдокод алгоритму

## Бульбашка

## temp, counter2 = 0, counter1=0;

ДЛЯ (i = 0; i < n - 1; i++)

ДЛЯ (j = 0; j < n - i - 1; j++)

counter1++;

Якщо (mas[j] > mas[j + 1])

То

counter2++;

temp = mas[j];

mas[j] = mas[j + 1];

mas[j + 1] = temp;

Гребінець

counter2 = 0, counter1 = 0;

gap = len;

swapped = 1;

Повторити Коли (gap > 1 та swapped)

swapped = false;

gap = newGap(gap);

Для (i = 0; i < len - gap; ++i)

counter1++;

якщо (mas[i] > mas[i + gap])

counter2++;

swap(mas[i], mas[i + gap]);

swapped = 1;

## Аналіз часової складності

Бульбашка:

n^2 – в найгіршому та випадковому випадку

n – в найкращому випадку

Гребінець:

n^2 – в найгіршому

n^2/2^p – в випадковому випадку(p – кі-сть виконання)

nlogn - в найкращому випадку

## Програмна реалізація алгоритму

### Вихідний код

#include<iostream>

using namespace std;

void inputBEST(int\* matr, int n);

void inputW(int\* mas, int n);

void inputR(int\* mas, int n);

void output(int\* matr, int n);

void sortB(int\* mas, int n);

void sortG(int\* mas, int len);

int main() {

cout << "input size" << endl;

int n;

cin >> n;

cout << endl;

int\* mas = new int[n];

cout << "the BEST" ;

inputBEST(mas, n);

sortB(mas, n);

cout << "the worst ";

inputW(mas, n);

sortB(mas, n);

cout << "random ";

inputR(mas, n);

sortB(mas, n);

//output(mas, n);

delete[] mas;

}

void inputBEST(int\* mas, int n) {

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

mas[i] = i;

}

}

void inputW(int\* mas, int n) {

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

mas[i] = n-i;

}

}

void inputR(int\* mas, int n) {

srand(time(NULL));

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

mas[i] = rand()%99999999;

}

}

void output(int\* mas, int n) {

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

cout << mas[i] << "\t";

}

}

void sortB(int\* mas, int n) {

int temp, counter2 = 0, counter1=0;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

counter1++;

if (mas[j] > mas[j + 1]) {

counter2++;

temp = mas[j];

mas[j] = mas[j + 1];

mas[j + 1] = temp;

}

}

}

cout << endl;

cout << "Number of comparisons = " << counter1 << endl;

cout << "Number of permutations = " << counter2 << endl;

cout << endl;

}

int newGap(int gap)

{

gap /= 1.247;

if (gap < 1)

return 1;

return gap;

}

void sortG(int\* mas, int len)

{

int counter2 = 0, counter1 = 0;

int gap = len;

bool swapped = true;

while (gap > 1 || swapped)

{

swapped = false;

gap = newGap(gap);

for (int i = 0; i < len - gap; ++i)

{

counter1++;

if (mas[i] > mas[i + gap])

{

counter2++;

swap(mas[i], mas[i + gap]);

swapped = true;

}

}

}

cout << endl;

cout << "Number of comparisons = " << counter1 << endl;

cout << "Number of permutations = " << counter2 << endl;

cout << endl;

}

### Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми сортування масивів на 100 і 1000 елементів відповідно.

Рисунок 3.1 – Сортування масиву на 100 елементів

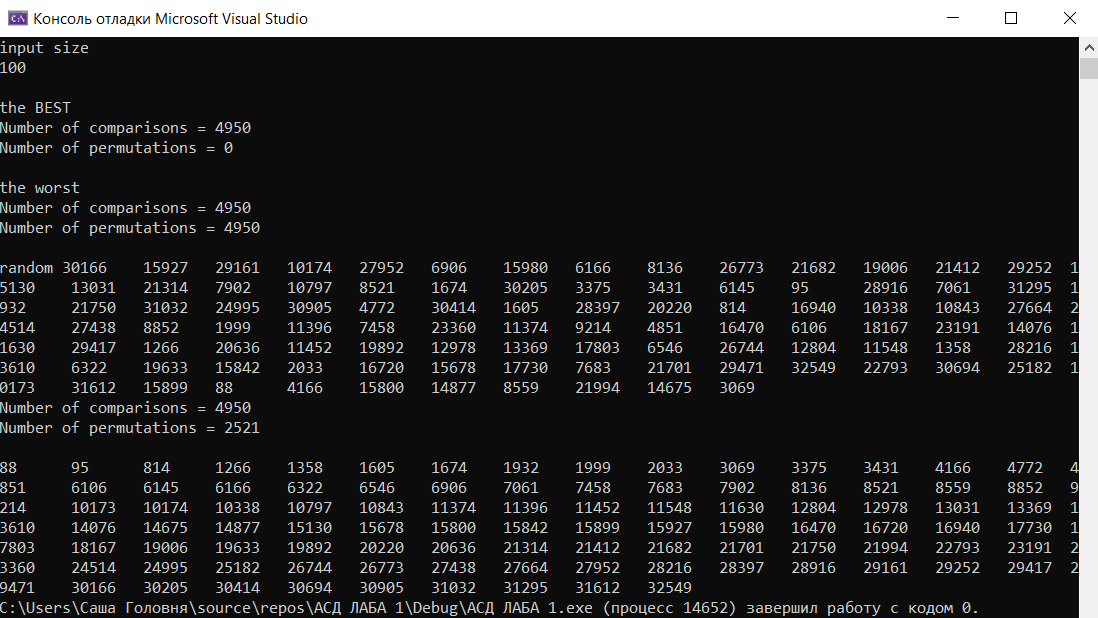


Рисунок 3.2 – Сортування масиву на 1000 елементів

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.2 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масив містить упорядковану послідовність елементів.

Таблиця 3.2 – Характеристики оцінювання алгоритму сортування бульбашки для упорядкованої послідовності елементів у масиві

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 0 |
| 100 | 4950 | 0 |
| 1000 | 499500 | 0 |
| 5000 | 12497500 | 0 |
| 10000 | 49995000 | 0 |
| 20000 | 199990000 | 0 |
| 50000 | 1249975000 | 0 |

ГРЕБІНЕЦЬ

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 36 | 0 |
| 100 | 1229 | 0 |
| 1000 | 22022 | 0 |
| 5000 | 144832 | 0 |
| 10000 | 329598 | 0 |
| 20000 | 719136 | 0 |
| 50000 | 1997680 | 0 |

В таблиці 3.3 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, коли масиви містять зворотно упорядковану послідовність елементів.

Таблиця 3.3 – Характеристики оцінювання алгоритму сортування бульбашки для зворотно упорядкованої послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 45 |
| 100 | 4950 | 4950 |
| 1000 | 499500 | 499500 |
| 5000 | 12497500 | 12497500 |
| 10000 | 49995000 | 49995000 |
| 20000 | 199990000 | 199990000 |
| 50000 | 1249975000 | 1249975000 |

ГРЕБІНЕЦЬ

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 9 |
| 100 | 1328 | 110 |
| 1000 | 23021 | 1512 |
| 5000 | 149831 | 9154 |
| 10000 | 339597 | 19018 |
| 20000 | 739135 | 40730 |
| 50000 | 2047679 | 110332 |

У таблиці 3.4 наведені характеристики оцінювання числа порівнянь і числа перестановок алгоритму сортування бульбашки для масивів різної розмірності, масиви містять випадкову послідовність елементів.

Таблиця 3.4 – Характеристика оцінювання алгоритму сортування бульбашки для випадкової послідовності елементів у масиві.

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 22 |
| 100 | 4950 | 2495 |
| 1000 | 499500 | 242122 |
| 5000 | 12497500 | 6238250 |
| 10000 | 49995000 | 24722446 |
| 20000 | 199990000 | 99823429 |
| 50000 | 1249975000 | 622091815 |

ГРЕБІНЕЦЬ

|  |  |  |
| --- | --- | --- |
| Розмірність масиву | Число порівнянь | Число перестановок |
| 10 | 45 | 7 |
| 100 | 1328 | 245 |
| 1000 | 23021 | 4305 |
| 5000 | 159829 | 27135 |
| 10000 | 349596 | 59032 |
| 20000 | 759134 | 126491 |
| 50000 | 2097678 | 345205 |

### Графіки залежності часових характеристик оцінювання від розмірності масиву

На рисунку 3.3 показані графіки залежності часових характеристик оцінювання від розмірності масиву для випадків, коли масиви містять упорядковану послідовність елементів (зелений графік), коли масиви містять зворотно упорядковану послідовність елементів (червоний графік), коли масиви містять випадкову послідовність елементів (синій графік), також показані асимптотичні оцінки гіршого (фіолетовий графік) і кращого (жовтий графік) випадків для порівняння.

Графіки залежності часових характеристик оцінювання

Рисунок 3.3 – Бульбашка

Висновок

При виконанні даної лабораторної роботи вивчили основні методи аналізу обчислювальної складності алгоритмів внутрішнього сортування і оцінити поріг їх ефективності. Заповнили таблицю порівнянь кількості перестановок та порівнянь для методів бульбашки та гребінця, подали відповідні графіки складності та залежності часу виконання та порівнянь від кількості елементів.

Критерії оцінювання

У випадку здачі лабораторної роботи до 21.02.2022 включно максимальний бал дорівнює – 5. Після 21.02.2022 – 28.02.2022 максимальний бал дорівнює – 2,5. Після 28.02.2022 робота не приймається

Критерії оцінювання у відсотках від максимального балу:

* аналіз алгоритму на відповідність властивостям – 10%;
* псевдокод алгоритму – 15%;
* аналіз часової складності – 25%;
* програмна реалізація алгоритму – 25%;
* тестування алгоритму – 20%;
* висновок – 5%.