Міністерство освіти і науки України Національний технічний університет "ХПІ" кафедра "Інформатика та інтелектуальна власність"

**РОЗРАХУНКОВО-** **ГРАФІЧНЕ ЗАВДАННЯ**

**(або КУРСОВА РОБОТА)**

з дисципліни "*Програмування"*

**Тема**

" Порівняльний аналіз методів сортування"

Виконав студент групи КН-320г

Шабаш Ігор Володимирович

Керівник РГЗ (або проекту)

Андрій Васильович Івашко

Харків 2020

Оглавление

[1 ДОКЛАДНИЙ ОПИС 3](#_Toc57663468)

[1.1 Сортування гребінцем опис 3](#_Toc57663469)

[1.2 Сортування гребінцем Псевдокод 3](#_Toc57663470)

[1.3 Швидке сортування хоара опис 4](#_Toc57663471)

[1.4 Швидке сортування Хоара Псевдокод 5](#_Toc57663472)

[2 РЕАЛІЗАЦІЯ МЕТОДІВ СОРТУВАННЯ МАСИВІВ 6](#_Toc57663473)

[2.1 Код сортування гребінцем 6](#_Toc57663474)

[2.2 Код швидкого сортування Хоара 7](#_Toc57663475)

[3 РЕАЛІЗАЦІЯ МЕТОДА АНАЛІЗУ ВИКОНАННЯ ПРОГРАММИ 8](#_Toc57663476)

[3.1 Код аналізу 8](#_Toc57663477)

[3.2 Опис аналізу 8](#_Toc57663478)

[4 ЕКСПЕРИМЕНТИ ВИЗНАЧЕННЯ ЧАСОВИХ ХАРАКТЕРИСТИК АЛГОРИТМІВ 9](#_Toc57663479)

[Таблиця 1 9](#_Toc57663480)

[Таблиця 2 9](#_Toc57663481)

[Таблиця 3 9](#_Toc57663482)

[5. ГРАФІК ЗАЛЕЖНОСТІ ЧАСУ 10](#_Toc57663483)

[5.1 Сортування гребінцем 10](#_Toc57663484)

[5.2 Швидке сортування Хоара 10](#_Toc57663485)

[6. ВИСНОВКИ 11](#_Toc57663486)

## 1 ДОКЛАДНИЙ ОПИС

### 1.1 Сортування гребінцем опис

Сортування гребінцем (англ. Comb sort) — спрощений алгоритм сортування, розроблений Влодеком Добошєвічем (Wlodek Dobosiewicz) у 1980 році, і пізніше заново дослідженим та популяризованим Стефаном Лакеєм (Stephen Lacey) та Річардом Боксом (Richard Box), котрі написали про нього в журналі Byte Magazine у квітні 1991 р. Сортування гребінцем є поліпшенням алгоритму сортування бульбашкою, і конкурує у швидкодії з алгоритмом Швидке сортування. Основна його ідея полягає в тому, щоб усунути так званих «черепах», або малих значень ближче до кінця списку, оскільки у сортування бульбашкою вони сильно уповільнюють процес сортування. (Кролики та великі сортування на початку списку у сортуванні бульбашкою не являють собою проблеми).

У сортуванні бульбашкою, коли два елементи порівнюються, вони завжди мають розрив (відстань один від одного) рівну 1. Основна ідея сортування гребінцем полягає у тому, що цей розрив може бути більший одиниці. (Алгоритм Сортування Шелла також базується на даній ідеї, однак, він є модифікацією алгоритму сортування включенням, а не сортування бульбашкою).

Розрив починається зі значення, що рівне довжині списку, поділеного на фактор зменшення, і список сортується з урахуванням цього значення (при необхідності воно заокруглюється до цілого). Потім розрив знову ділиться на фактор розриву, і список продовжує сортуватись з новим значенням, процес продовжується доти, доки розрив рівний 1. Далі список сортується з розривом рівним 1 доти, доки не буде повністю відсортований. Таким чином, фінальний етап сортування аналогічний такому ж у сортуванні бульбашкою, однак, до цього «черепахи» усуваються.

### 1.2 Сортування гребінцем Псевдокод

функція Сортування гребінцем (введення масиву)

проміжок: = вхідний.розмір

виконувати доки проміжок <= 1 і кількість свапів = 0

якщо проміжок > 1

проміжок: = проміжок / 1.3

якщо проміжок = 10 або проміжок = 9

розрив: = 11

закінчити, якщо

закінчити, якщо

i: = 0

свапи: = 0

виконувати доки i + проміжок <= вхідний.розмір

якщо input [i]> input [i + проміжок]

обмін (input [i], input [i + проміжок])

свапи: = 1

закінчити, якщо

i: = i + 1

кінцева закінчити виконання

кінець закінчити виконання

кінець функція

### 1.3 Швидке сортування хоара опис

Швидке сортування (англ. Quick Sort) — алгоритм сортування, розроблений Тоні Гоаром, який не потребує додаткової пам'яті і виконує у середньому O(nlog nоперацій. Однак, у найгіршому випадку робить O(n^2) порівнянь. Позаяк алгоритм використовує дуже прості цикли і операції, він працює швидше за інші алгоритми, що мають таку ж асимптотичну оцінку складності. Наприклад, зазвичай більш ніж удвічі швидший порівняно з сортуванням злиттям.

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як у масиві, так і в двозв'язному списку.

Дана схема використовує два індексу (один на початку масиву, інший в кінці), які наближаються одна до одної, поки не знайдеться пара елементів, де один більше опорного і розташований перед ним, а другий менше і розташований після. Ці елементи міняються місцями. Обмін відбувається до тих пір, поки індекси не перетнуться. Алгоритм повертає останній індекс. Схема Хоара ефективніше схеми Ломуто, так як відбувається в середньому в три рази менше обмінів (swap) елементів, і розбиття ефективніше, навіть коли всі елементи рівні. Подібно схемою Ломуто, дана схема також показує ефективність в O (n^2), коли вхідний масив вже відсортований. Сортування з використанням даної схеми нестабільна. Слід зауважити, що кінцева позиція опорного елемента необов'язково збігається з повернутими індексом

### 1.4 Швидке сортування Хоара Псевдокод

'''algorithm''' quicksort(A, lo, hi) '''is'''

'''if''' lo < hi then

p:= partition(A, lo, hi)

quicksort(A, lo, p)

quicksort(A, p + 1, hi)

'''algorithm''' partition(A, low, high) '''is'''

pivot:= A[(low + high) / 2]

i:= low

j:= high

'''loop forever'''

'''while''' A[i] < pivot

i:= i + 1

'''while''' A[j] > pivot

j:= j - 1

'''if''' i >= j '''then'''

'''return''' j

'''swap''' A[i] '''with''' A[j]

## 2 РЕАЛІЗАЦІЯ МЕТОДІВ СОРТУВАННЯ МАСИВІВ

### 2.1 Код сортування гребінцем

// Найти зазор между элементами

int getNextGap(int gap)

{

// Усадочный зазор по коэффициенту

gap = (gap \* 10) / 13;

if (gap < 1)

return 1;

return gap;

}

// Функция для сортировки [0..n-1] с помощью Comb Sort

void combSort(int a[], int n)

{

int gap = n;

bool swapped = true;

// Продолжаем работать, пока пробел больше 1 и последний

// итерация вызвала обмен

while (gap != 1 || swapped == true)

{

// Находим следующий пробел

gap = getNextGap(gap);

// Инициализируем замену как false, чтобы мы могли

// проверяем, произошел ли своп или нет

swapped = false;

// Сравнить все элементы с текущим разрывом

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

{

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

{

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

swapped = true;

}

}

}

}

### 2.2 Код швидкого сортування Хоара

int partition(int arr[], int low, int high)

{

int pivot = arr[low];

int i = low - 1, j = high + 1;

while (true) {

// Находим крайний левый элемент больше, чем

// или равно pivot

do {

i++;

} while (arr[i] < pivot);

// Находим крайний правый элемент меньше, чем

// или равно pivot

do {

j--;

} while (arr[j] > pivot);

// Если встретились два указателя.

if (i >= j)

return j;

swap(arr[i], arr[j]);

}

}

/\*Основная функция, реализующая QuickSort

arr[]->Массив для сортировки,

низкий->Начальный индекс,

высокий->Индекс окончания\*/

void quickSort(int arr[], int low, int high)

{

if (low < high) {

/\*pi - индекс разделения, теперь arr[p]

в нужном месте\*/

int pi = partition(arr, low, high);

// Отдельно сортировать элементы перед

// разделoм и после раздела

quickSort(arr, low, pi);

quickSort(arr, pi + 1, high);

}

}

## 3 РЕАЛІЗАЦІЯ МЕТОДА АНАЛІЗУ ВИКОНАННЯ ПРОГРАММИ

### 3.1 Код аналізу

#include <iostream>

#include <chrono>

using namespace std;

using namespace std::chrono;

#define N 5

//Код сортування

int main()

{

int a[N] ;

//for (int i = 0; i < N; a[i++] = i);//Упорядоченный массив

for (int i = 0; i < N; a[i++] = N-i);//Массив, упорядоченный в обратном порядке

//for (int i = 0; i < N; a[i++] = rand() % 201 - 100);//Неупорядоченный массив

auto start = high\_resolution\_clock::now();

combSort(a, N);

//quickSort(a, 0, N - 1);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" << endl;

return 0;}

### 3.2 Опис аналізу

Тут мы використовували бібліотеку chrono для визначення часу виконання программи.

Запуск auto start = high\_resolution\_clock::now();

Алгоритм сортування

Кінець auto stop = high\_resolution\_clock::now();

Підрахунок auto duration = duration\_cast<microseconds>(stop - start);

## 4 ЕКСПЕРИМЕНТИ ВИЗНАЧЕННЯ ЧАСОВИХ ХАРАКТЕРИСТИК АЛГОРИТМІВ

### Таблиця 1

Упорядкованный массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| СОРТУВАННЯ 1 | 2 | 6 | 7 | 6 | 6 | 7 | 7 | 6 | 7 |
| СОРТУВАННЯ 2 | 2 | 6 | 6 | 7 | 7 | 8 | 9 | 10 | 11 |

### Таблиця 2

Упорядкованный массив в обратному порядку

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| СОРТУВАННЯ 1 | 7 | 6 | 6 | 8 | 8 | 9 | 9 | 12 | 10 |
| СОРТУВАННЯ 2 | 7 | 7 | 8 | 9 | 9 | 10 | 10 | 10 | 12 |

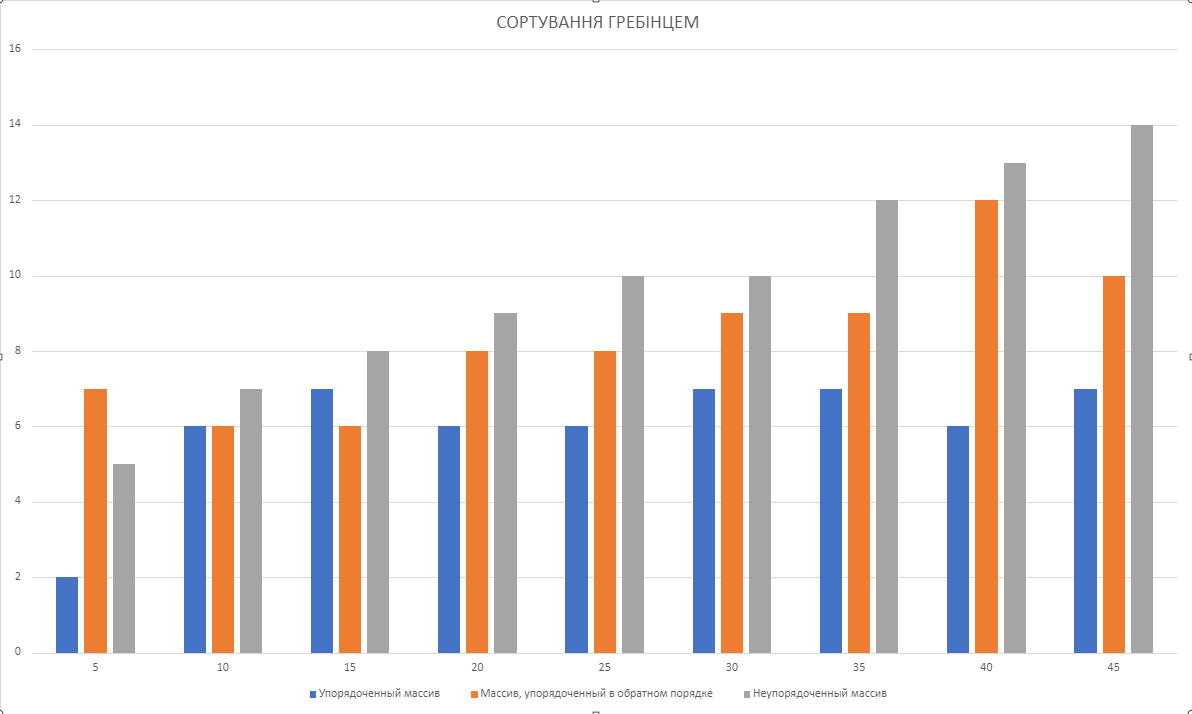
### Таблиця 3

Упорядкованный массив

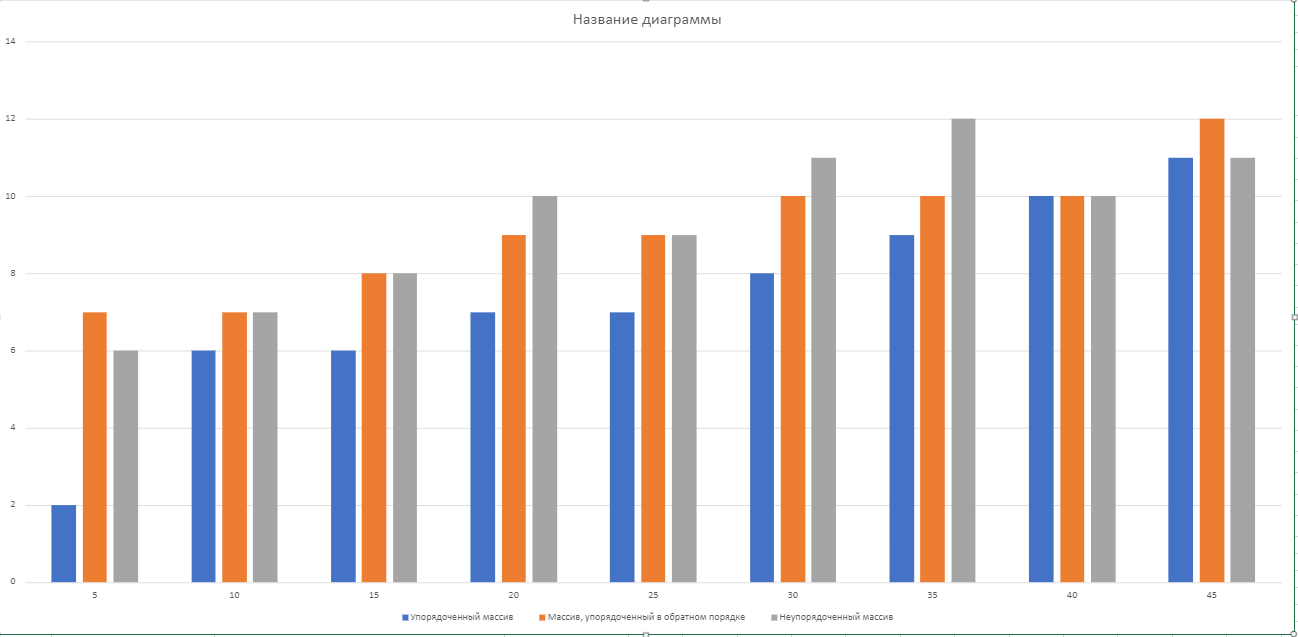
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| СОРТУВАННЯ 1 | 5 | 7 | 8 | 9 | 10 | 10 | 12 | 13 | 14 |
| СОРТУВАННЯ 2 | 6 | 7 | 8 | 10 | 9 | 11 | 12 | 10 | 11 |

## 5. ГРАФІК ЗАЛЕЖНОСТІ ЧАСУ

### 5.1 Сортування гребінцем



### 5.2 Швидке сортування Хоара



## 6. ВИСНОВКИ

Вході виконання роботи було дослідженно два алгоритми сортування, а саме Сортування гребінцем та Швидке сортування Хоара. Експерементально було дослідженно, що алгоритм Сортування гребінцем працює швидше лише на малих розмірах массива, коли швидке сортування Хоара більш ефективне на великих розмірах массивів. З цього робимо висновок, що Сортування гребінцем слід використовувата з невеликими обсяягами інформаціі, а Швидке сортування Хоара – з великими.