 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Лабораторна робота №6**

з дисципліни «Алгоритмізація та програмування

на тему: «Алгоритми сортування»

Варіант №

**Виконав:**

**Перевірив:**

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2022

**Завдання:**

1. Ознайомитися з теоретичними відомостями створення масивів, розглянувши алгоритми бульбашкового, коктейльного сортування, сортування гребінцем, швидкого сортування, сортування простими вставками, сортування Шелла, злиттям, вибором та пірамідального сортування. .

2. Відповідно до свого варіанту розробки програмний застосунок, що включає створення масиву цілих чисел, заповнення його елементів випадковими згенерованими значеннями з діапазону **[0, 50]**, сортування і форматований вивід на екран початкового та відсортованого масиву (глобальні змінні не використовувати, розміри матриці задавати з клавіатури, функція *main* має бути призначена тільки для виклику функцій користувача):

**Завдання за варіантом:** мається одновимірний масив довжиною ***N=40***. Відсортувати за зменшенням алгоритмом сортування злиттям ті елементи масиву, які є парними числами.

3. Скласти і захистити звіт по роботі.

**Програмна реалізація:**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

void generate\_arr(int\*, int);

void mergeSort(int\*, int, int);

void merge(int\*, int, int, int);

void arr\_copy\_even(int\*, int\*, int);

void arr\_out(int\*, int);

int calc\_even(int\*, int);

void ex2();

void ex3();

int main()

{

int choice;

cout << "What do you want to do (ex2 = 0, ex3 = 1): ";

cin >> choice;

if (choice)

ex3();

else

ex2();

return 0;

}

void ex2()

{

int len = 0;

cout << "Enter length of the array: ";

cin >> len;

int\* arr = new int[len];

generate\_arr(arr, len);

cout << "Unsorted array:\n";

arr\_out(arr, len);

int len\_even = calc\_even(arr, len);

int\* arr\_even = new int[len\_even];

arr\_copy\_even(arr, arr\_even, len);

cout << "\nEvens:\n";

arr\_out(arr\_even, len\_even);

mergeSort(arr\_even, 0, len\_even - 1);

cout << "\nSorted array:\n";

arr\_out(arr\_even, len\_even);

delete[] arr;

delete[] arr\_even;

}

void ex3()

{

int c;

cout << "How many arrays you want to test: ";

cin >> c;

for (c; c > 0; c--)

{

int len;

cout << "Length of the array: ";

cin >> len;

int\* arr = new int[len];

generate\_arr(arr, len);

unsigned int start\_time = clock();

int len\_even = calc\_even(arr, len);

int\* arr\_even = new int[len\_even];

arr\_copy\_even(arr, arr\_even, len);

mergeSort(arr\_even, 0, len\_even - 1);

unsigned int end\_time = clock();

cout << "Time to sort the array with length of " << len << " = " << (end\_time - start\_time) << " milliseconds" << endl;

delete[] arr;

delete[] arr\_even;

}

}

void generate\_arr(int\* arr, int n)

{

srand(static\_cast<unsigned int>(time(0)));

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

{

arr[i] = int(rand() % 51);

}

}

void merge(int arr[], int left\_ind, int mid\_ind, int right\_ind)

{

int len1 = mid\_ind - left\_ind + 1;

int len2 = right\_ind - mid\_ind;

int\* L = new int[len1];

int\* M = new int[len2];

for (int i = 0; i < len1; i++)

L[i] = arr[left\_ind + i];

for (int j = 0; j < len2; j++)

M[j] = arr[mid\_ind + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = left\_ind;

while (i < len1 && j < len2)

{

if (L[i] >= M[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

while (i < len1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < len2)

{

arr[k] = M[j];

j++;

k++;

}

delete[] M;

delete[] L;

}

void mergeSort(int\* arr, int left\_ind, int right\_ind)

{

if (left\_ind < right\_ind)

{

int mid\_ind = left\_ind + (right\_ind - left\_ind) / 2;

mergeSort(arr, left\_ind, mid\_ind);

mergeSort(arr, mid\_ind + 1, right\_ind);

merge(arr, left\_ind, mid\_ind, right\_ind);

}

}

void arr\_out(int\* arr, int len)

{

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

cout << arr[i] << '\t';

}

int calc\_even(int\* arr1, int len)

{

int num = 0;

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

{

if (arr1[i] % 2 == 0)

num++;

}

return num;

}

void arr\_copy\_even(int\* arr1, int\* arr2, int len)

{

int j = 0;

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

{

if (arr1[i] % 2 == 0)

{

arr2[j] = arr1[i];

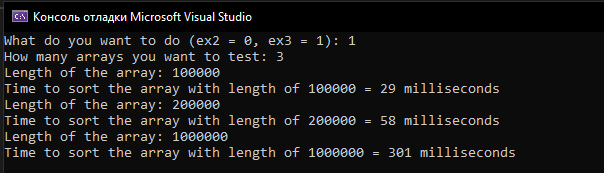
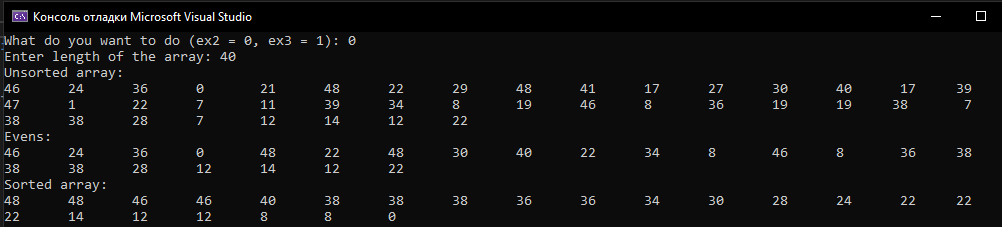
j++;

}

}

}

**Результат виконання програми:**

****

**Контрольні питання:**

1. Що таке сортування? В чому його основна мета?

Сортування - це процес перегрупування заданої множини об’єктів, що приводить до їх впорядкованого розташування щодо ключа.

Мета сортування - полегшити подальший пошук елементів.

1. Чим можна пояснити різноманітність алгоритмів сортування?

Універсального, найкращого алгоритму сортування на даний момент не

існує. Сортувати можна одні і ті ж дані різними способами.

1. Опишіть сутність критеріїв, за якими оцінюються алгоритми сортування.

Оцінка алгоритмів сортування

проводиться за наступними параметрами:

- час сортування (параметр, що характеризує швидкодію алгоритму);

- пам’ять (параметр, який характеризується тим, що ряд алгоритмів

сортування вимагають виділення додаткової пам’яті під тимчасове

зберігання даних);

- стійкість (параметр, який відповідає за те, що сортування не змінює

взаємного розташування рівних елементів);

- природність поведінки (параметр, який вказує на ефективність методу

при обробці вже відсортованих або частково відсортованих даних;

алгоритм поводиться природно, якщо враховує цю характеристику

вхідної послідовності, і працює краще).

1. Чому не існує універсального алгоритму сортування?

Бо кожен алгоритм має якісь переваги і недоліки, по відношенню до інших алгоритмів сортування.

1. Які характеристики даних треба враховувати при виборі алгоритму сортування?

Кількість елементів, їх складність та кількість пам’яті, яку вони займають.

1. Чи впливає тип даних, що сортуються, на швидкість алгоритму та на час сортування?

Впливає. Можлива ситуація, коли для того, щоб відсортувати дані, треба дістати ключ, тобто виконати певні операції, які також займають час. Також є важливим розмір даних, але це можна не враховувати у випадку, коли у алгоритмі переміщуються не самі елементи, а вказівники на них, але порівняння їх усе одно займає різну кількість часу, бо можливо порівняння в 1 процесорну команду (невеликий розмір елемента) і в декілька процесорних команд (великий розмір елемента).

1. Які на сьогоднішній день найефективніші алгоритми сортування?

Quick sort, сортування вставками, сортування Шелла.

1. За рахунок чого в швидких алгоритмах сортування досягається перевага у швидкості?

Швидкість досягається за рахунок того, що при сортуванні, на певному кроці такі алгоритми починають порівнювати елементи не всі з усіма, а тільки з частиною елементів. Також швидкі алгоритми використовують додаткові ресурси.

1. Чому алгоритми швидкого сортування не дають великого виграшу часу на малих кількостях елементів?

Недоліком алгоритму є його рекурсивність, яка викликає необхідність витрачати пам’ять комп’ютера на запис адрес повернення з підпрограми сортування кожного з підмасивів. Цей недолік і є причиною того, що на малих кількостях елементів, алгоритм працюватиме гірше.

1. Чим відрізняється внутрішнє сортування від зовнішнього?

Внутрішнє сортування - працює з даними в оперативній пам’яті з

довільним доступом.Зовнішнє сортування - впорядковує інформацію, розташовану на

зовнішніх носіях.