Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Сравнение сортировок.

Выполнил:

студент института ИТММ

гр. 3821Б1ПМ3

Мартынов А.Ю.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород

2021г.

# Содержание

[Содержание 1](#_Toc90682505)

[Введение 2](#_Toc90682506)

[Постановка задачи 3](#_Toc90682507)

[Руководство пользователя 4](#_Toc90682508)

[Первая программа 4](#_Toc90682509)

[Вторая программа 6](#_Toc90682510)

[Руководство программиста 8](#_Toc90682511)

[*Описание структуры кода программ* 8](#_Toc90682512)

[*Первая программа* 8](#_Toc90682513)

[Вторая программа 10](#_Toc90682514)

[Описание алгоритмов 13](#_Toc90682515)

[Сортировка «пузырьком» 13](#_Toc90682516)

[Сортировка «вставками» 15](#_Toc90682517)

[Эксперименты 18](#_Toc90682518)

[Заключение 19](#_Toc90682519)

[Список литературы 19](#_Toc90682520)

[Приложение 1 20](#_Toc90682521)

[Приложение 1.1 20](#_Toc90682522)

[Приложение 2 22](#_Toc90682523)

[Приложение 2.1 22](#_Toc90682524)

[Приложение 2.2 24](#_Toc90682525)

# Введение

Очень часто в решении прикладных задач для определённых алгоритмов требуется использование упорядоченных векторов – массивов чисел, например, при использовании бинарного поиска, который в сочетании с быстрой сортировкой может стать хорошим инструментом для поиска чисел в массиве. Если же данных много, то сталкиваемся с проблемой обработки этих данных – сортировкой. Чтобы ускорить этот процесс, были придуманы специальные алгоритмы. Их довольно много, в своей лабораторной работе я буду рассматривать 3 вида сортировок: пузырьковая, вставками, быстрая. Реализуем эти сортировки, сравним их и сделаем выводы.

# Постановка задачи

Реализовать сортировки массивов данных (тип данных определяется преподавателем(float)) задаваемых: обязательно случайно.

Реализовать сортировки: пузырьком, вставкой, быстрая. Сравнить время работы, сделать выводы.

Первая программа должна создавать текстовый файл с записанными в него числами. Программа принимает количество чисел n, максимальное и минимальное значение.

Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия.

# Руководство пользователя

## Первая программа

После запуска программы выходит сообщение о выборе заполнении файла – случайными числами или же ввод чисел с клавиатуры (рис. 1)



Рис.1 Запуск программы

Затем пользователь должен ввести минимальное и максимальное число и количество чисел. После генерации и записи в файл будет выведено сообщение «done» (рис. 2).

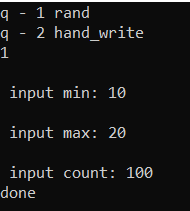


Рис. 2 Генерация массива

Если же пользователь хочет ввести числа с клавиатуры, то тогда в самом первом выводе сообщения на консоль, он должен напечатать цифру 2. Потом ему надо будет ввести количество этих чисел, а затем и сами числа. (Рис 3)

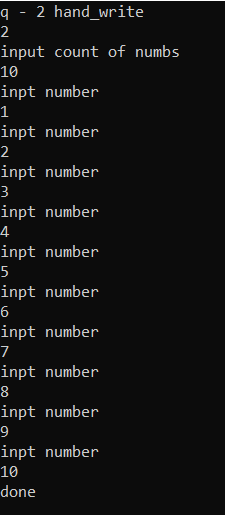


Рис. 3 – ручной ввод чисел

## Вторая программа

При запуске второй программы, перед пользователем появляется меню, в котором написаны возможные действия. Для того чтобы выбрать какое-то действие (рис. 4).

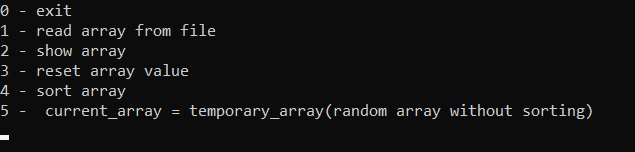


Рис.4 Меню программы

Если пользователь введёт число 0, то программа завершится. Если же пользователь введёт 1, то тогда программа считает массив из файла и запишет его в память. Если пользователь введёт 2, то на экран выведется массив. Для увеличения скорости работы элементы массива выводятся через пробел, тк при достаточно большом количестве элементов, вывод 1 элемента на 1 строчку будет занимать куда больше времени(рис.5).

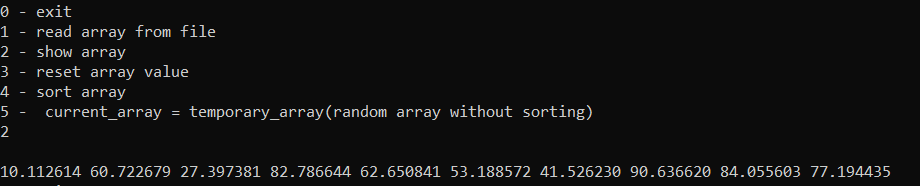


Рис. 5 Вывод массива

Если пользователь напишет в консоли число 4, то программа предложит выбрать следующие сортировки: пузырьком, вставками и быстрая. Пользователю необходимо выбрать нужную сортировку (рис. 6).

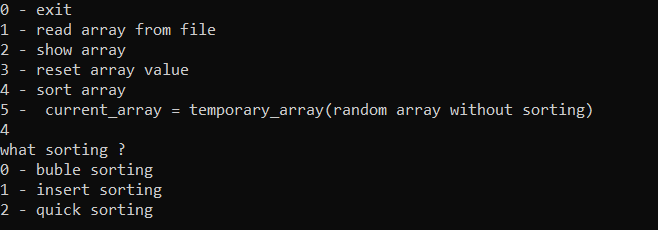


Рис. 6 Выбор сортировки

Далее пользователь должен выбрать вариант сортировки опять же введя число. Затем выйдет соответствующее сообщение о времени выбранной сортировки. (рис. 7)

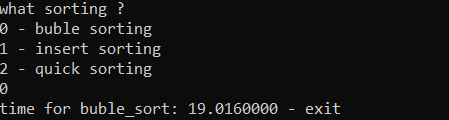


Рис. 7 Время сортировки

Для того, чтобы отсортированный массив вернуть к изначальному виду, нужно ввести цифру 5. Если надо обнулить массив, то тогда нужно ввести цифру 3.

Для выхода из программы нужно ввести цифру 0. (рис.8)

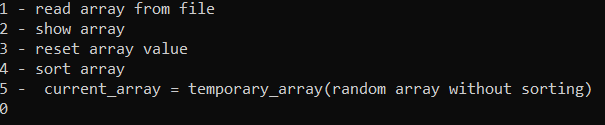


рис.8. Выход из меню

# Руководство программиста

# *Описание структуры кода программ*

# *Первая программа*

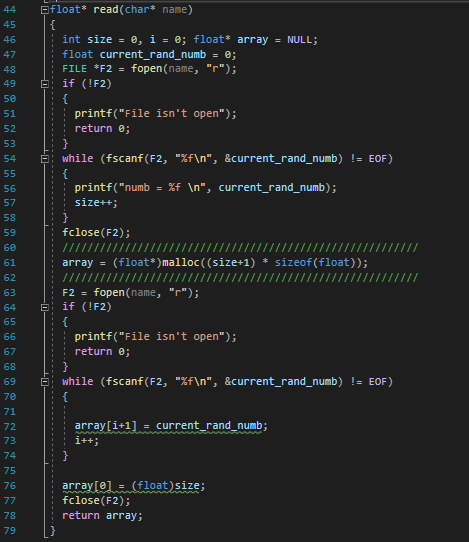
В первой программе используются следующие библиотеки: stdio.h, stdlib.h – для работы с файлами (Рис 9). Программа состоит из нескольких функций: FILE\* write(char\* name)- функция заполнения файла случайными числами(Рис. 10), float\* read(char\* name) – функция получения массива, состоящего из чисел, которые взяты из файла, причём 0 элемент массива – это количество чисел в данном массиве(Рис. 11), FILE\* hand\_write(char\* name) – функция ручного ввода массива(Рис. 12), int main(void) – тело программы(Рис. 13)



Рис 9 – библиотеки



Рис 10 – функция заполнения случайными числами

 Рис. 11 - функция получения массива из файла

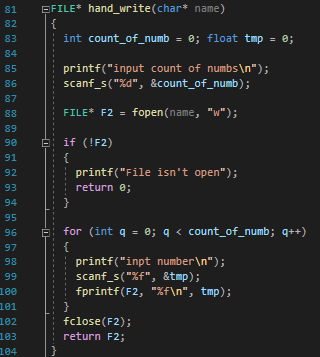
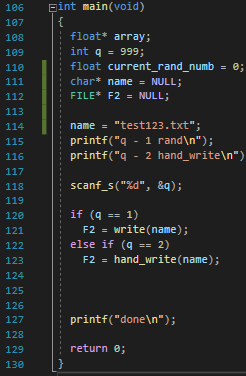
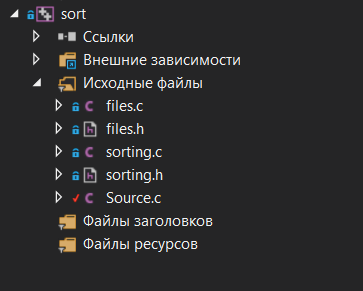


Рис. 12 – функция ввода чисел с клавиатуры и запись их в файл

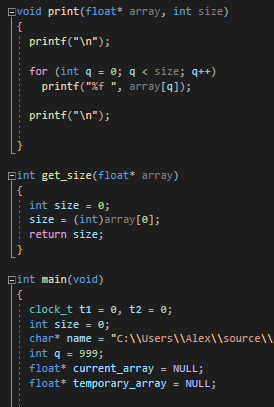
 Рис. 13 – тело программы

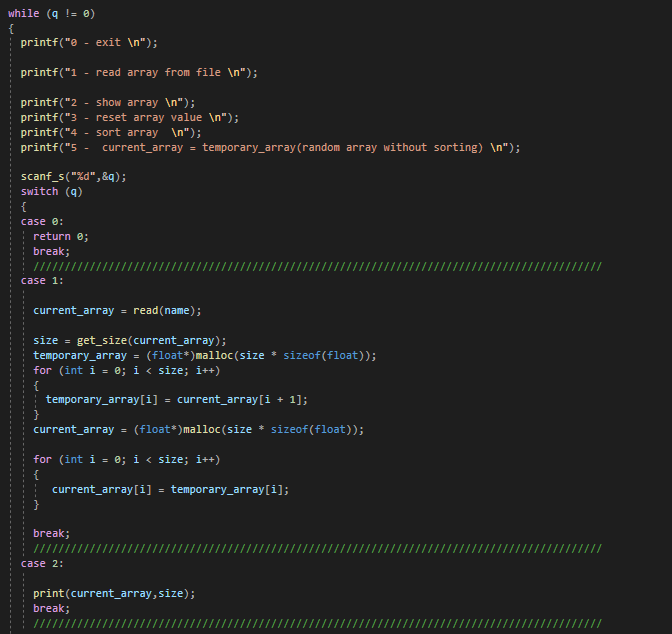
### Вторая программа

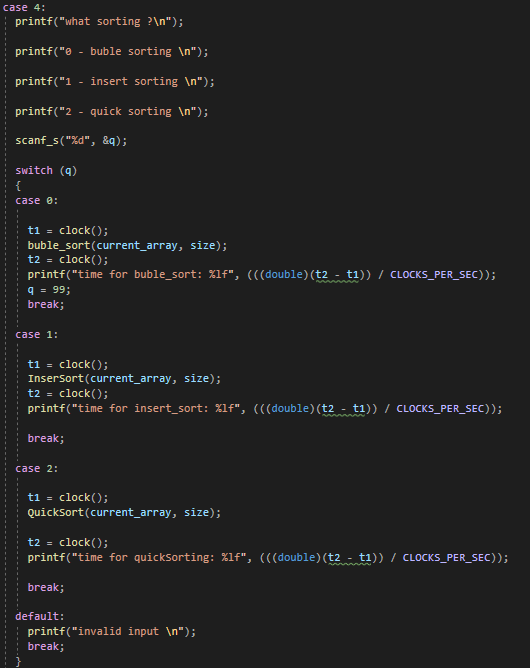
Вторая программа состоит из 5-и файлов (Рис. 14): source.c – функция main(), files.c – это копия первой программы, но без функции main(), чтобы была возможность использовать функции для работы с файлами в файле source.c. Файл files.h – заголовочный файл, подключаемый в файле source.c, для возможности использования функции для работы с файлами. Файл sorting.c – это файл реализации сортировок: пузырьковая, вставками и быстрая сортировки. Подключается в файл source.c через заголовочный файл sorting.h

 Рис.14 – структура второй программы

Рассмотрим подробнее файлы sorting.c и source.c (файл files.c был рассмотрен выше) Программа состоит из 3-х функций (Рис. 15) – print (float\* array, int size). Данная функция печатает на экран массив размера size. Функция get\_size(float\* array) возвращает размер массива, полученного при чтении файла. Функция main() (Рис. 16)в основном состоит из оператора switch – case, где в цикле while(1) перебираются все варианты. Главным модулем в реализации задачи является модуль вызова сортировок и подсчёта времени (Рис. 17)

 Рис. 15 Структура файла source.c

 Рис. 16 – Структура функции main()

 Рис. 17 – реализация главного модуля подсчёта времени сортировок

## Описание алгоритмов

### Сортировка «пузырьком»

Каждый проход начинается с начала последовательности. Сравнивается первый элемент со вторым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются второй с третьим, третий с четвертым и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный элемент будет находится на последнем месте в массиве (Рис. 18), (Блок-схема №1)

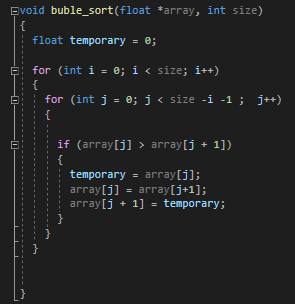


Рис.18 – реализация сортировки пузырьком



Блок-схема №1 – Сортировка пузырьком

### Сортировка «вставками»

  Первый элемент в массиве образует уже отсортированную последовательность. Сравниваем второй элемент с первым. Если порядок между ними нарушен, то первый элемент передвигается на одну позицию вправо. Теперь отсортированный массив состоит из двух элементов.

Далее, в течение каждой итерации, берем следующий элемент (третий, четвертый и т.д) и сравниваем его поочередно с другими элементами в уже отсортированном списке, **начиная с конца** этого списка. Если порядок между сравниваемыми элементами нарушен, то меняем их местами, если нет, то “вставка” нового элемента закончена, переходим к следующему(Рис.19), (блок-схема №2)

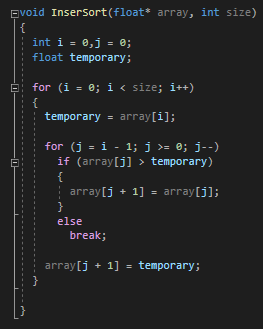
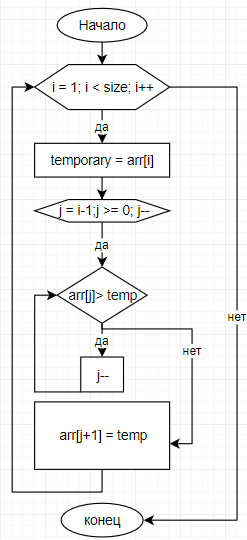


Рис. 19 – реализация сортировки вставками



Блок-схема №2 – описание алгоритма сортировки вставками

***Быстрая сортировка***

в отличии от двух других сортировок алгоритм быстрой сортировки работает по совершенно другому принципу. Для начала выбирается средний по положению элемент входного массива – ключ. Затем просматриваются элементы массива до ключа и после него. Как только слева находится элемент больше ключа, и справа находится элемент меньше ключа, то они меняются местами. Это происходит до тек пор, пока все элементы слева от ключа не будут меньше него, а все справа не станут больше.

Таким образом мы имеем два массива: массив с числами меньше ключа и массив с числами больше ключа. Затем происходит рекурсивный вызов сортировки для каждого из этих массивов. Этот процесс происходит до тех пор, пока все элементы массива не окажутся отсортированными, то есть пока один из массивов на некоторой итерации не станет одноэлементным(Рис. 20)

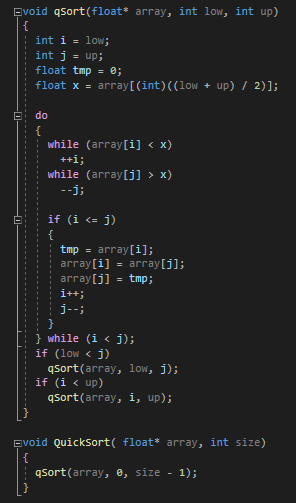


Рис. 20 – реализация быстрой сортировки

Данный алгоритм принципиально отличается от двух других. Это касается и его сложности. В сравнении с квадратичными сортировками (пузырьковая, вставками), быстрая сортировка работает за O (n \* log(n), в то время, как пузырьковая и вставками работает за O (n\*n)

# Эксперименты

Сравним разные виды сортировок при помощи программы. Для этого создадим для них одинаковый массив чисел, который будет использоваться 3 раза. Замерим время на режиме debug, т.к. в режиме release компилятор может ускорять код, что отрицательно скажется на результатах эксперимента. Замерим каждой сортировки и внесём в таблицу №1. Все замеры проводятся для чисел типа float в диапазоне от 10 до 320000

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип сортировки** | **Сложность** | **Размер входных данных** | **Время работы, с** |
| Пузырьком |  | 5000 | ≈0,063 |
| Вставками |  | 5000 | ≈0,02 |
| Быстрая |  | 5000 | ≈0,001 |
| Пузырьком |  | 100000 | ≈35,624 |
| Вставками |  | 100000 | ≈9,664 |
| Быстрая |  | 100000 | ≈0,02 |

Таблица 1. Сравнение времени работы сортировок

# Заключение

В данной лабораторной работе я написал 2 проекты. Отработал работу с файлами, а именно чтение из файла, запись чисел в файл. Отработал алгоритмы разных сортировок, сравнил скорость их работы в программе, попрактиковался в работе с указателями и динамическими массивами.

# Список литературы

1. C. Полное руководство. Герберт Шилдт Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с.

# Приложение 1

## Приложение 1.1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

FILE\* write(char\* name)

{

int min = 0, max = 0, count\_of\_numb = 0;

float current\_rand\_numb = 0;

printf("\n input min: ");

scanf\_s("%d", &min);

printf("\n input max: ");

scanf\_s("%d", &max);

printf("\n input count: ");

scanf\_s("%d", &count\_of\_numb);

if (min > max)

{

printf("\n min > max - err ");

return 0;

}

FILE\* F2 = fopen(name, "w");

if (!F2)

{

printf("File isn't open");

exit(0);

}

for (int i = 0; i < count\_of\_numb; i++)

{

current\_rand\_numb = (((float)rand()) / RAND\_MAX) \* ((float)max - (float)min) + min;

fprintf(F2, "%f\n", current\_rand\_numb);

}

fclose(F2);

return F2;

}

float\* read(char\* name)

{

int size = 0, i = 0; float\* array = NULL;

float current\_rand\_numb = 0;

FILE \*F2 = fopen(name, "r");

if (!F2)

{

printf("File isn't open");

return 0;

}

while (fscanf(F2, "%f\n", &current\_rand\_numb) != EOF)

{

printf("numb = %f \n", current\_rand\_numb);

size++;

}

fclose(F2);

/////////////////////////////////////////////////////////

array = (float\*)malloc((size+1) \* sizeof(float));

/////////////////////////////////////////////////////////

F2 = fopen(name, "r");

if (!F2)

{

printf("File isn't open");

return 0;

}

while (fscanf(F2, "%f\n", &current\_rand\_numb) != EOF)

{

array[i+1] = current\_rand\_numb;

i++;

}

array[0] = (float)size;

fclose(F2);

return array;

}

FILE\* hand\_write(char\* name)

{

int count\_of\_numb = 0; float tmp = 0;

printf("input count of numbs\n");

scanf\_s("%d", &count\_of\_numb);

FILE\* F2 = fopen(name, "w");

if (!F2)

{

printf("File isn't open");

return 0;

}

for (int q = 0; q < count\_of\_numb; q++)

{

printf("inpt number\n");

scanf\_s("%f", &tmp);

fprintf(F2, "%f\n", tmp);

}

fclose(F2);

return F2;

}

int main(void)

{

float\* array;

int q = 999;

float current\_rand\_numb = 0;

char\* name = NULL;

FILE\* F2 = NULL;

name = "test123.txt";

printf("q - 1 rand\n");

printf("q - 2 hand\_write\n");

scanf\_s("%d", &q);

if (q == 1)

F2 = write(name);

else if (q == 2)

F2 = hand\_write(name);

printf("done\n");

return 0;

}

# Приложение 2

## Приложение 2.1

#include<stdio.h>

#include<stdlib.h>

void buble\_sort(float \*array, int size)

{

float temporary = 0;

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

{

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

{

if (array[j] > array[j + 1])

{

temporary = array[j];

array[j] = array[j+1];

array[j + 1] = temporary;

}

}

}

}

void InserSort(float\* array, int size)

{

int i = 0,j = 0;

float temporary;

for (i = 0; i < size; i++)

{

temporary = array[i];

for (j = i - 1; j >= 0; j--)

if (array[j] > temporary)

{

array[j + 1] = array[j];

}

else

break;

array[j + 1] = temporary;

}

}

void qSort(float\* array, int low, int up)

{

int i = low;

int j = up;

float tmp = 0;

float x = array[(int)((low + up) / 2)];

do

{

while (array[i] < x)

++i;

while (array[j] > x)

--j;

if (i <= j)

{

tmp = array[i];

array[i] = array[j];

array[j] = tmp;

i++;

j--;

}

} while (i < j);

if (low < j)

qSort(array, low, j);

if (i < up)

qSort(array, i, up);

}

void QuickSort( float\* array, int size)

{

qSort(array, 0, size - 1);

}

## Приложение 2.2

#include<stdio.h>

#include<stdlib.h>

#include"files.h"

#include"sorting.h"

#include<time.h>

#include<math.h>

void print(float\* array, int size)

{

printf("\n");

for (int q = 0; q < size; q++)

printf("%f ", array[q]);

printf("\n");

}

int get\_size(float\* array)

{

int size = 0;

size = (int)array[0];

return size;

}

int main(void)

{

clock\_t t1 = 0, t2 = 0;

int size = 0;

char\* name = "C:\\Users\\Alex\\source\\repos\\second\_lab\_1\_c\\second\_lab\_1\_c\\test123.txt";

int q = 999;

float\* current\_array = NULL;

float\* temporary\_array = NULL;

while (q != 0)

{

printf("0 - exit \n");

printf("1 - read array from file \n");

printf("2 - show array \n");

printf("3 - reset array value \n");

printf("4 - sort array \n");

printf("5 - current\_array = temporary\_array(random array without sorting) \n");

scanf\_s("%d",&q);

switch (q)

{

case 0:

return 0;

break;

///////////////////////////////////////////////////////////////////////////////////////////

case 1:

current\_array = read(name);

size = get\_size(current\_array);

temporary\_array = (float\*)malloc(size \* sizeof(float));

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

{

temporary\_array[i] = current\_array[i + 1];

}

current\_array = (float\*)malloc(size \* sizeof(float));

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

{

current\_array[i] = temporary\_array[i];

}

break;

///////////////////////////////////////////////////////////////////////////////////////////

case 2:

print(current\_array,size);

break;

///////////////////////////////////////////////////////////////////////////////////////////

case 3:

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

{

current\_array[i] = 0;

}

break;

case 4:

printf("what sorting ?\n");

printf("0 - buble sorting \n");

printf("1 - insert sorting \n");

printf("2 - quick sorting \n");

scanf\_s("%d", &q);

switch (q)

{

case 0:

t1 = clock();

buble\_sort(current\_array, size);

t2 = clock();

printf("time for buble\_sort: %lf", (((double)(t2 - t1)) / CLOCKS\_PER\_SEC));

q = 99;

break;

case 1:

t1 = clock();

InserSort(current\_array, size);

t2 = clock();

printf("time for insert\_sort: %lf", (((double)(t2 - t1)) / CLOCKS\_PER\_SEC));

break;

case 2:

t1 = clock();

QuickSort(current\_array, size);

t2 = clock();

printf("time for quickSorting: %lf", (((double)(t2 - t1)) / CLOCKS\_PER\_SEC));

break;

default:

printf("invalid input \n");

break;

}

break;

///////////////////////////////////////////////////////////////////////////////////////////

case 5:

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

{

current\_array[i] = temporary\_array[i];

}

break;

default:

printf("invalid input \n");

}

}

return 0;

}