**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: «Программирование и исследование алгоритмов сортировки»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6382 |  | Воропаев А.О. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2017

**Цель работы.**

Реализовать выбранный алгоритм сортировки и провести экспериментальное испытание (исследование) алгоритма (программной реализации).

**Постановка задачи.**

Вариант 2: cортировка вставками (простые вставки (просмотр назад/вперед), бинарные вставки)

Требуется:

Провести экспериментальное испытание алгоритма сортировки вставками. При этом оценить эффективность алгоритма: время выполнения ( с использованием компьютерного таймера) и количество произведенных базовых операций алгоритма – операций сравнения и операций присваивания (перемещения элементов). По результатам испытаний необходимо проанализировать:

1) зависимость времени выполнения, количества сравнений и количества присваиваний от размера массива,

2) соотношение измеренных и теоретически предсказанных значений,

3) сравнительные характеристики эффективности разных вариаций

(модификаций) алгоритмов, и др., в зависимости от конкретного задания.

Во всех случаях рекомендуется провести сравнения (по тем показателям, по которым это возможно) с каким-либо «эталонным» алгоритмом (реализацией), например, с системной функцией qsort или с какой-либо ещё библиотечной функцией.

**Описание алгоритма.**

Рассмотрим алгоритм метода сортировки вставками на примере сортировки по возрастанию. 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 течении 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.

Сортировка вставками имеет большую вычислительную сложность. Поэтому она эффективна на небольших наборах данных. Рекомендуется использовать этот метод на наборах размером до десятков элементов. Сортировка вставками эффективна на последовательностях с данными, которые уже частично отсортированы.

Бинарные вставки отличаются от простых вставок тем,  
что место вставки ищется двоичным поиском, вместо последовательного поиска.

**Спецификация программы.**

Ограничения на входные данные: входные данные - массив целых чисел.

Место и форма представления входных данных: входные данные генерируются программой случайно (фиксирован лишь размер массива), либо вводятся из файла, либо вводятся с консоли, представляют собой набор целых чисел.

Выходные данные: программа выводит на консоль отсортированный массив и промежуточные этапы сортировки, а также в отдельный файл только отсортированный массив (для анализа результата).

**Алгоритм работы программы.**

В данной лабораторной работе нужно отсортировать массив чисел простыми и бинарными вставками.

Для начала программа получает входные данные (способы получения данных указаны в предыдущем пункте) и перед началом работы записывает их в файл input, который больше изменяться не будет (для последующего анализа корректности алгоритма). Пользователь вводит размер массива.

После этого пользователь выбирает опции работы с массивом (Отсортировать простыми вставками с подсчетом времени, отсортировать бинарными вставками с подсчетом времени, отсортировать функцией qsort, отсортировать простыми вставками с подсчетом количества операций, отсортировать бинарными вставками с подсчетом количества операций).

Затем программа начинает выполнение своей основной задачи - сортировки (алгоритм показан в пункте “Описание алгоритма”). В это время на консоль выводятся промежуточные состояния массива после каждого шага сортировки. Если размер массива больше 16 элементов, то промежуточные результаты выводится не будут.

После завершения сортировки результаты можно посмотреть в файле sorted.txt

**Описание основных функций.**

В программе используется 10 функций помимо основной - int main(): void RandGenFunc(int \*array, long size), void printArr(int \*array, int size), void BinaryInsertionSortTimeCount(int\* array, long size, FILE \*f), void BinaryInsertionSortOperationsCount(int\* array, long size, FILE \*f), void InsertionSortTimeCount(int \*array, long size, FILE \*f), void InsertionSortOperationsCount(int \*array, long size, FILE \*f), int cmp(const void \* x1, const void \* x2), void QsortOption(int \*array, int size), void ReadFromFile(int \*array, long size), void ConsoleRead(int \*array, int size). Рассмотрим каждую подробнее.

1. void RandGenFunc(int \*array, long size)

На вход функция получает указатель на сортируемый массив и его размер. В процессе работы генерирует в массив случайные числа.

2. void printArr(int \*array, int size)

На вход функции подается указатель на массив и его размер. Функция выводит на экран все элементы массива последовательно.

3. void BinaryInsertionSortTimeCount(int\* array, long size, FILE \*f)

На вход функция получает указатели на массив и на файл, в который будет произведена запись отсортированного массива, а также количество элементов массива. По ходу работы функции будет произведена сортировка бинарными вставками. Место вставки каждого элемента будет выбираться с помощью бинарного поиска. Функция также считает время, затраченное на выполнение сортировки в секундах (если размер массива больше 16 элементов). Если размер массива меньше 16 элементов, то сортировка будет произведена с выводом всех промежуточных состояний массива. В конце отсортированный массив будет записан в файл sorted.txt

4. void BinaryInsertionSortOperationsCount(int\* array, long size, FILE \*f)

На вход функция получает указатели на массив и на файл, в который будет произведена запись отсортированного массива, а также количество элементов массива. По ходу работы функции будет произведена сортировка бинарными вставками. Место вставки каждого элемента будет выбираться с помощью бинарного поиска. Функция также считает количество элементарных операций (если размер массива больше 16 элементов). Если размер массива меньше 16 элементов, то сортировка будет произведена с выводом всех промежуточных состояний массива. В конце отсортированный массив будет записан в файл sorted.txt

5. void InsertionSortTimeCount(int \*array, long size, FILE \*f)

На вход функция получает указатели на массив и на файл, в который будет произведена запись отсортированного массива, а также количество элементов массива. По ходу работы функции будет произведена сортировка простыми вставками. Функция также считает время, затраченное на выполнение сортировки в секундах (если размер массива больше 16 элементов). Если размер массива меньше 16 элементов, то сортировка будет произведена с выводом всех промежуточных состояний массива. В конце отсортированный массив будет записан в файл sorted.txt

6. void InsertionSortOperationsCount(int \*array, long size, FILE \*f)

На вход функция получает указатели на массив и на файл, в который будет произведена запись отсортированного массива, а также количество элементов массива. По ходу работы функции будет произведена сортировка простыми вставками. Функция также считает количество элементарных операций (если размер массива больше 16 элементов). Если размер массива меньше 16 элементов, то сортировка будет произведена с выводом всех промежуточных состояний массива. В конце отсортированный массив будет записан в файл sorted.txt

7. int cmp(const void \* x1, const void \* x2)

Вспомогательная функция, необходимая для применения функции qsort на вход принимает два значения и возвращает 1, если первое значение больше, 0 – если значения равны и -1 – если больше второе значение.

8. void QsortOption(int \*array, int size)

Функция принимает на вход указатель на массив и количество элементов массива. В процессе работы сортирует массив с помощью библиотечной функции qsort.

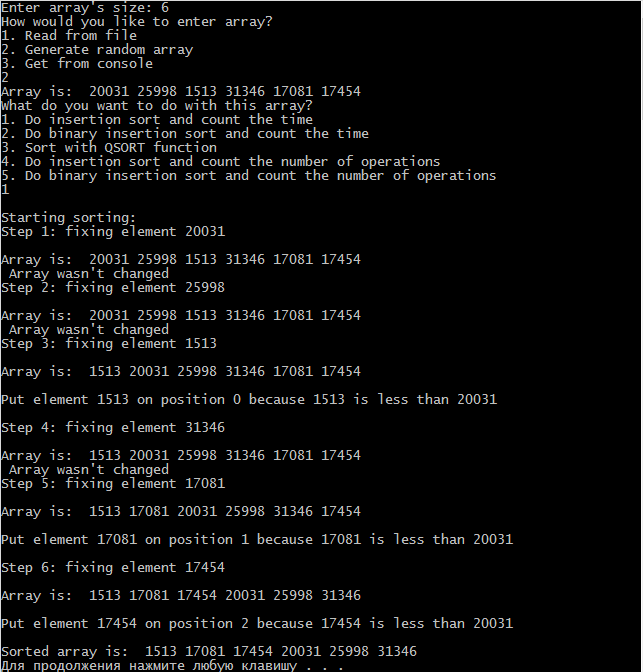
9. void ReadFromFile(int \*array, long size)

Функция принимает на вход указатель на массив и количество элементов массива. В процессе работы считывает из файла массив. Если количество элементов в файле меньше, чем должно быть в массиве, то остальные элементы будут сгенерированы случайно.

10. void ConsoleRead(int \*array, int size)

Функция принимает на вход указатель на массив и количество элементов массива. В процессе работы позволяет пользователю ввести массив с клавиатуры.

**Пример работы программы.**



**Тестирование программы.**

Рассмотрим тесты на корректных входных данных для подсчета количества элементарных операций.

|  |  |  |
| --- | --- | --- |
| **Неотсортированный массив** | **Отсортированный массив** | **Дополнительные данные** |
| -72 -81 -95 -62 -78 -56 -88 | -95 -88 -81 -78 -72 -62 -56 | Простые вставки:  Присваивания: 24  Сравнения: 10  Бинарные вставки:  Присваивания: 32  Сравнения: 20 |
| -9 15 -19 -9 -31 -35 22 -39 28 19 -13 -27 -21 13 25 -46 -45 -1 25 49 -23 11 12 -33 29 11 -28 -37 -1 21 11 -42 31 17 30 -3 33 38 -20 -38 24 28 -17 -27 8 33 26 9 27 35 -25 -32 48 -34 50 29 38 -14 3 22 45 43 -11 28 -45 -35 6 -24 12 15 7 19 13 -19 13 18 -4 -10 -4 -35 37 -1 -3 -33 11 -42 -8 -11 14 36 -1 -7 -6 -3 19 -25 17 8 7 -15 | -46 -45 -45 -42 -42 -39 -38 -37 -35 -35 -35 -34 -33 -33 -32 -31 -28 -27 -27 -25 -25 -24 -23 -21 -20 -19 -19 -17 -15 -14 -13 -11 -11 -10 -9 -9 -8 -7 -6 -4 -4 -3 -3 -3 -1 -1 -1 -1 3 6 7 7 8 8 9 11 11 11 11 12 12 13 13 13 14 15 15 17 17 18 19 19 19 21 22 22 24 25 25 26 27 28 28 28 29 29 30 31 33 33 35 36 37 38 38 43 45 48 49 50 | Простые вставки:  Присваивания: 2592  Сравнения: 2392  Бинарные вставки:  Присваивания: 3650  Сравнения: 1123 |
| 16806 17005 17825 15174 14531 14256 17443 15108 12060 17006 12092 16633 16062 15619 16508 18763 18214 10076 16511 18260 | 10076 12060 12092 14256 14531 15108 15174 15619 16062 16508 16511 16633 16806 17005 17006 17443 17825 18214 18260 18763 | Простые вставки:  Присваивания: 128  Сравнения: 88  Бинарные вставки:  Присваивания: 226  Сравнения: 125 |
| 8 5 | 5 8 | Простые вставки:  Присваивания: 5  Сравнения: 1  Бинарные вставки:  Присваивания: 5  Сравнения: 3 |

Примечание: при каждой проверке на корректных данных сортировка программы сходилась с сортировкой функции qsort.

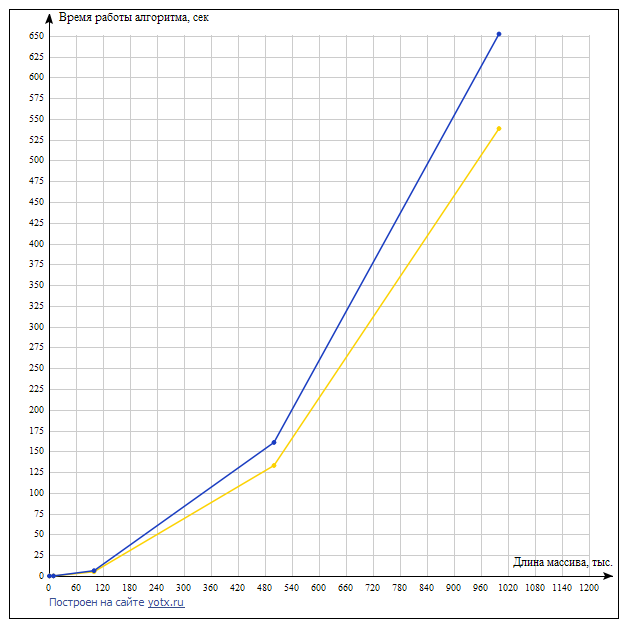
**Исследование.**

В данной лаборатнорной работе реализовывался алгоритм сортировки вставками. Для начала рассмотрим зависимость времени выполнения (см. рис. 1), количества сравнений (см. рис. 2) и количества присваиваний (см. рис. 3) от размера массива для сортировки простыми вставками.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер массива | Время, сек. | Сравнения | Присваивания | Время qsort, сек. |
| 100 | Близко к нулю | 2392 | 2592 | Близко к нулю |
| 1 000 | 0,001 | 238457 | 240457 | 0,001 |
| 10 000 | 0,07 | 25027426 | 25047426 | 0,006 |
| 100 000 | 6,353 | 2497197092 | 2497397092 | 0,076 |
| 500 000 | 160,699 | 62569014476 | 62570014476 | 0,326 |
| 1 000 000 | 652,167 | 250276057904 | 250280057904 | 0,632 |

Теперь рассмотрим те же случаи для сортировки бинарными вставками

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер массива | Время, сек. | Сравнения | Присваивания | Время qsort, сек. |
| 100 | Близко к нулю | 620 | 3827 | Близко к нулю |
| 1 000 | 0,001 | 9569 | 272089 | 0,001 |
| 10 000 | 0,064 | 128888 | 25215940 | 0,006 |
| 100 000 | 5,104 | 1622375 | 2497225203 | 0,076 |
| 500 000 | 132,959 | 9273897 | 62640082224 | 0,326 |
| 1 000 000 | 538,393 | 37095588 | 250560328896 | 0,632 |



* сортировка простыми вставками
* сортировка бинарными вставками

Рисунок 1 - График зависимости времени сортировки от длины массива

Теперь построим график зависимости количества присваиваний от длины массива. Так как разница в количестве присваиваний для бинарной сортировки и для сортировки простыми вставками незначительна (для массивов размерами более 100 тыс. элементов разница составляет менее 1%), то отразим количество присваиваний на одном графике для двух видов сортировок.

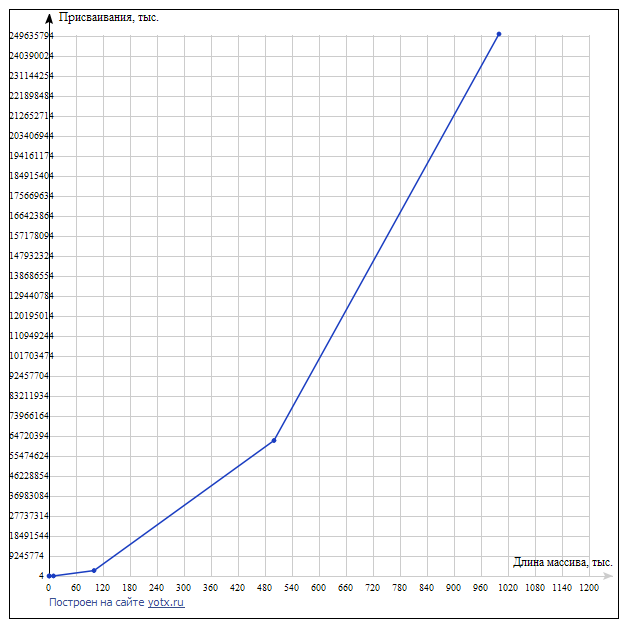


Рисунок 2 - График зависимости количества присваиваний от длины массива

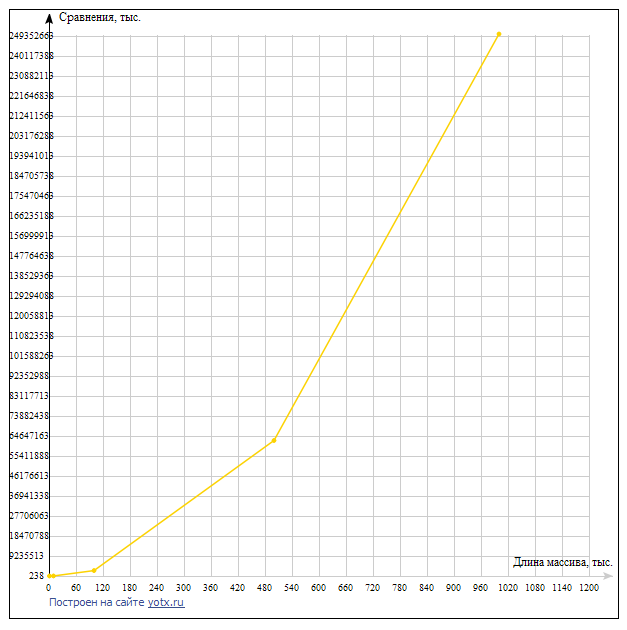


Рисунок 3 - График зависимости количества сравнений от длины массива для простых вставок

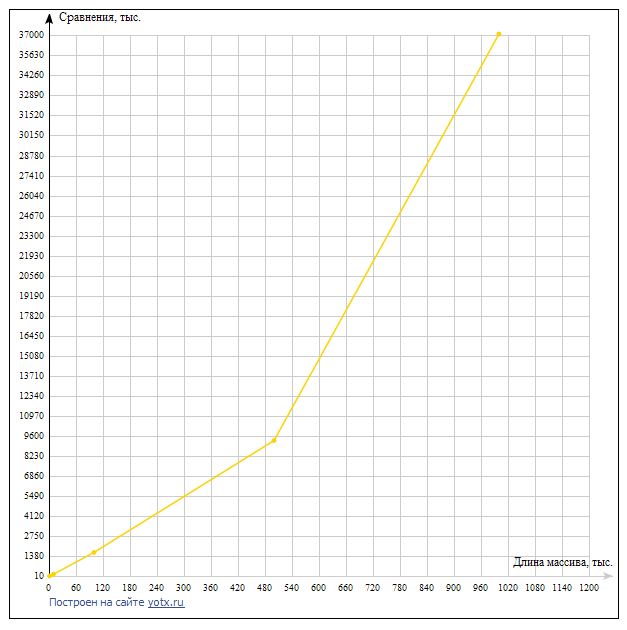


Рисунок 4 – График зависимости количества сравнений от длины массива для бинарных вставок

Как мы можем наблюдать, количество сравнений при сортировке простыми вставками приблизительно в 1000 раз больше, чем при сортировке бинарными вставками. Именно из-за этого сортировка бинарными вставками выгодней сортировки простыми вставками на контейнерах с большим количеством элементов.

Однако, сравнивая обе эти сортировки с библиотечной функцией qsort, можем заметить, что они приблизительно одинаково эффективны на массивах с менее, чем 10000 элементов. При большем же числе элементов сортировка qsort показывает намного более выгодный результат, как и ожидалось до проведения эксперимента. Это подтверждает и теоретическая сложность алгоритмов O(n2) – для сортировки вставками и O(nlogn) – для qsort.

**Вывод.**

В результате выполнения этой лабораторной работы была исследована сортировка массива простыми и бинарными вставками, найдена зависимость количества базовых операций и времени от длины сортируемого массива и проведено сравнение с библиотечной функцией сортировки qsort.

**Приложение.**

**Исходный код программы.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <ctime>

using namespace std;

void RandGenFunc(int \*array, long size)

{

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

{

array[i] = rand(); //генерирует случайное число

}

}

void printArr(int \*array, int size)

{

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

{

cout << " " << array[i]; //печатает элементы массива

}

cout << endl;

}

void BinaryInsertionSortTimeCount(int\* array, long size, FILE \*f)

{

int x, temp=1;

long left,j;

long right;

long sred;

if (size <= 16)//если размер массива меньше 16 элементов

{

cout << endl << "Starting sorting: " << endl;

cout << "Step 1: fixing element " << array[0] << endl << endl;

cout << "Array is: ";

printArr(array, size);

cout << endl << "Array wasn't changed " << endl << endl;

}

unsigned int start\_time = clock();

for (long i = 1; i < size; i++)

{

if (size <= 16)

{

cout << "Step " << temp + 1 << ": fixing element " << array[i] << endl << endl;

cout << "Array is: ";

printArr(array, size);

}

temp++;

if (array[i - 1] > array[i])

{

x = array[i];

left = 0;

right = i - 1;

do {

sred = (left + right) / 2;//выбираем средний элемент

if (array[sred] < x) left = sred + 1;//если х больше среднего, то переходим в правую от среднего часть

else right = sred - 1;// иначе - в левую

} while (left <= right);

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

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

array[left] = x;// вставляем элемент на нужную позицию

if (size <= 16)

{

cout << endl << "Put element " << x << " on position " << j + 1 << " because " << x << " is less than " << array[j + 2] << endl;

cout << endl << "Array after this step is: ";

printArr(array, size);

cout << "\n";

}

}

else if (size <= 16)

cout << endl << "Array wasn't changed" << endl << endl;;

}

if (size > 16)

{

unsigned int end\_time = clock();

unsigned int time = end\_time - start\_time;

cout << "Work time is " << (float)time / CLOCKS\_PER\_SEC << " sec" << endl;//выводим время

}

else

{

cout << "Sorted array is: ";

printArr(array, size);

}

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

{

fprintf(f, "%d ", array[i]);//выводим массив в файл

}

}

void BinaryInsertionSortOperationsCount(int\* array, long size, FILE \*f)

{

int x, temp = 1;

long long comparisons = 0, assignments = 0,j;//кол-во присваиваний и сравнений

long left;

long right;

long sred;

if (size <= 16)

{

cout << endl << "Starting sorting: " << endl;

cout << "Step 1: fixing element " << array[0] << endl << endl;

cout << "Array is: ";

printArr(array, size);

cout << endl << "Array wasn't changed " << endl << endl;

}

for (long i = 1; i < size; i++)

{

if (size <= 16)

{

cout << "Step " << temp + 1 << ": fixing element " << array[i] << endl << endl;

cout << "Array is: ";

printArr(array, size);

}

temp++;

comparisons++;//увеличиваем кол-во присваиваний

if (array[i - 1] > array[i])

{

x = array[i]; // Основной алгоритм такой же, как и в функции, считающей

left = 0; // время, только считаются операции

right = i - 1;

assignments+=3;

do {

sred = (left + right) / 2;

assignments++;//увеличиваем счетчик присваиваний

comparisons++;//увеличиваем счетчик сравнений

if (array[sred] < x)

{

left = sred + 1;

assignments++;

}

else

{

right = sred - 1;

assignments++;

}

} while (left <= right);

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

{

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

assignments++;

}

array[left] = x;

assignments++;

if (size <= 16)

{

cout << endl << "Put element " << x << " on position " << j + 1 << " because " << x << " is less than " << array[j + 2] << endl;

cout << endl << "Array after this step is: ";

printArr(array, size);

cout << "\n";

}

}

else if (size <= 16)

cout << endl << "Array wasn't changed" << endl << endl;

}

if (size<=16)

{

cout << "Sorted array is: ";

printArr(array, size);

}

//......................Выводим кол-во присваиваний и сравнений

cout << "The number of comparisons is: " << comparisons << endl << "The number of assignments is: " << assignments << endl;

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

{

fprintf(f, "%d ", array[i]);//записываем в файл

}

}

void InsertionSortTimeCount(int \*array, long size, FILE \*f)

{

int x, temp=1;

long i, j;

if(size<=16)

cout << endl << "Starting sorting: " << endl;

unsigned int start\_time = clock();//время начала

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

{

x = array[i];

if(size<=16)

cout << "Step " << temp <<": fixing element " << x << endl<<endl;

for (j = i - 1; j >= 0 && array[j] > x; j--)//ищем место для вставки элемента

{

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

}

array[j + 1] = x;//вставляем элемент на нужное место

if (size <= 16)

{

cout << "Array is: ";

printArr(array, size);

if ((j + 1) == i)

{

cout << " Array wasn't changed";

}

else

{

cout << endl << "Put element " << x << " on position " << j+1 << " because " << x << " is less than " << array[j+2] << endl;

}

cout << "\n";

temp++;

}

}

if (size > 16)

{

unsigned int end\_time = clock();//время конца

unsigned int time = end\_time - start\_time;

cout << "Work time is " << (float)time/CLOCKS\_PER\_SEC << " sec" << endl;//выводим время

}

else

{

cout << "Sorted array is: ";

printArr(array, size);

}

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

{

fprintf(f, "%d ", array[i]);// записываем в файл

}

}

void InsertionSortOperationsCount(int \*array, long size, FILE \*f)

{

int x, temp = 1;

long long comparisons = 0, assignments = 0;

long i, j;

if (size <= 16)

cout << endl << "Starting sorting: " << endl;

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

{

x = array[i];

if (size <= 16)

cout << "Step " << temp << ": fixing element " << x << endl << endl;

assignments++;

for (j = i - 1; j >= 0 && array[j] > x; j--)

{

comparisons++;

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

assignments++;

}

array[j + 1] = x;

assignments++;

if (size <= 16)

{

cout << "Array is: ";

printArr(array, size);

if ((j + 1) == i)

{

cout <<endl<< "Array wasn't changed"<<endl;

}

else

{

cout << endl << "Put element " << x << " on position " << j + 1 << " because " << x << " is less than " << array[j + 2] << endl;

}

cout << "\n";

temp++;

}

}

if(size<=16)

{

cout << "Sorted array is: ";

printArr(array, size);

cout << endl;

}

cout << "The number of comparisons is: " << comparisons << endl << "The number of assignments is: " << assignments << endl;

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

{

fprintf(f, "%d ", array[i]);

}

}

int cmp(const void \* x1, const void \* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

}

void QsortOption(int \*array, int size)

{

unsigned int start\_time = clock();

qsort(array, size, sizeof(int), cmp); //сортируем функцией qsort

unsigned int end\_time = clock();

unsigned int time = end\_time - start\_time;

cout << "Time of qsort is " << (float)time/CLOCKS\_PER\_SEC<<endl;// выводим время

}

void ReadFromFile(int \*array, long size)

{

FILE \*f;

long i = 0;

fopen\_s(&f, "input.txt", "r");//открываем файл

while (!feof(f))

{

fscanf\_s(f, "%d", &array[i]);//считываем оттуда элементы массива

i++;

}

if (i < size - 1)

{

srand(time(0));

RandGenFunc(&array[i], size - i);//если в файле элементов меньше, чем нужно, то генерируем остальные случайно

}

fclose(f);//закрываем файл

}

void ConsoleRead(int \*array, int size)

{

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

{

cout << "Enter the " << i + 1 << " element: "; //считываем с консоли

cin >> array[i];

}

}

int main()

{

FILE \*out;

fopen\_s(&out, "sorted.txt", "wt");//открываем файл для вывода

long size;

int cases;

printf("Enter array's size: ");

cin >> size; //вводим размер массива

int \*array = new int[size];

int choice;

cout << "How would you like to enter array?\n1. Read from file\n2. Generate random array\n3. Get from console\n";

do

{

cin >> choice;

if (choice <1 || choice >3)

cout << endl << "Try again\n";

} while (choice < 1 || choice > 3); //выбор опции

switch (choice)

{

case 1:

ReadFromFile(array, size); //чтение из файла

break;

case 2:

srand(time(0));

RandGenFunc(array, size); //генерация случайных чисел

break;

case 3:

ConsoleRead(array, size); //ввод с консоли

break;

}

if (size <= 16)

{

cout << "Array is: ";

printArr(array, size); //выводим массив

}

cout << "What do you want to do with this array?\n1. Do insertion sort and count the time"

"\n2. Do binary insertion sort and count the time\n3. Sort with QSORT function\n"

"4. Do insertion sort and count the number of operations\n5. Do binary insertion sort and count the number of operations"<<endl;

do

{

cin >> cases;

} while (cases<1 || cases >5);

switch (cases)

{

case 1:

InsertionSortTimeCount(array, size, out); //сортировка простыми вставками с подсчетом времени

break;

case 2:

BinaryInsertionSortTimeCount(array, size, out);// сортировка бинарными вставками с подсчетом времени

break;

case 3:

QsortOption(array, size); //сортировка qsort'ом

break;

case 4:

InsertionSortOperationsCount(array, size, out); //сортировка простыми вставками с подсчетом операций

break;

case 5:

BinaryInsertionSortOperationsCount(array, size, out);// сортировка бинарными вставками с подсчетом операций

break;

default:

break;

}

fclose(out); //закрыли выходной файл

return 0;

}