МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**МЕТОДЫ ПОИСКА И СОРТИРОВКИ**

**Выполнил:** студент группы

3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.А.Ермолаев

Подпись

**Проверил:** к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.А. Баркалов

Подпись

Нижний Новгород  
2022

**Содержание**

[1.ВВЕДЕНИЕ 3](#_Toc120457752)

[2.ПОСТАНОВКА ЗАДАЧИ 4](#_Toc120457753)

[3.ОПИСАНИЕ АЛГОРИТМОВ 5](#_Toc120457754)

[3.1 Пузырьковая сортировка 5](#_Toc120457755)

[3.2 Сортировка выбором 5](#_Toc120457756)

[3.3 Сортировка вставками 5](#_Toc120457757)

[3.4 Сортировка подсчетом 6](#_Toc120457758)

[3.5 Быстрая сортировка 6](#_Toc120457759)

[3.6 Сортировка слиянием 6](#_Toc120457760)

[3.7 Линейный поиск 6](#_Toc120457761)

[3.8 Бинарный поиск 6](#_Toc120457762)

[4.ОПИСАНИЕ ПРОГРАММЫ 7](#_Toc120457763)

[4.1 Описание глобальных переменных. 7](#_Toc120457764)

[4.2 Описание функций. 7](#_Toc120457765)

[5.РЕЗУЛЬТАТЫ 11](#_Toc120457766)

[6.ЗАКЛЮЧЕНИЕ 12](#_Toc120457767)

[7.ЛИТЕРАТУРА 13](#_Toc120457768)

[8.ПРИЛОЖЕНИЕ 14](#_Toc120457769)

# 1.ВВЕДЕНИЕ

Зачем нужны сортировки? В первую очередь для поиска и предоставления данных. Некоторые задачи с неотсортированными данными решить очень трудно, а некоторые просто невозможно.

Что и какого типа мы будем сортировать? Мы будем сортировать массив целых чисел типа int, используя сравнение и обмен. При подсчёте сравнений, таковым будем считать только сравнение между элементами массива.

Также в работе реализованы алгоритмы поиска: линейный и бинарный.

# 2.ПОСТАНОВКА ЗАДАЧИ

Изучить популярные алгоритмы сортировки целых чисел и реализовать их с помощью языка программирования C, а также сравнить работу алгоритмов на значительных массивах данных.

# 3.ОПИСАНИЕ АЛГОРИТМОВ

## 3.1 Пузырьковая сортировка

### 3.1.1 Наивная реализация

Идём по массиву от начала до конца, меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место встанет максимальный элемент. Снова обходим неотсортированную часть массива (от первого элемента до N-1 элемента) и меняем неотсортированных соседей. Второй по величине элемент окажется на n-1 месте. Продолжая, будем обходить уменьшающуюся неотсортированную часть массива, перемещая найденные максимумы в конец.

### 3.1.2 Реализация с прерыванием

Основана на наивной реализации, но с добавлением флага. Если во время прохода по массиву обменов не совершилось, то значит массив отсортирован, флаг срабатывает, и сортировка завершается.

### 3.1.3 Шейкер-сортировка

Перемещаем максимум в самый конец. После этого разворачиваемся и идём в обратную сторону, при этом уже перемещая в начало не максимум, а минимум. Отсортировав в массиве первый и последний элементы, снова делаем разворот. Обойдя туда-обратно несколько раз, в итоге заканчиваем процесс, оказавшись в середине списка.

## 3.2 Сортировка выбором

Ищем локальный минимум в неотсортированной части массив. Меняем местами локальный минимум и первый элемент неотсортированного массива. Повторяем, пока неотсортированный массив содержит два и более элемента

## 3.3 Сортировка вставками

### 3.3.1 Сортировка вставками с линейным поиском

Проходим по массиву слева направо и обрабатываем по очереди каждый элемент. Слева от очередного элемента наращивается отсортированная часть массива, справа по мере процесса уменьшается неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента с помощью линейного поиска. Сам элемент отправляется в буфер, в результате чего в массиве появляется свободная ячейка — это позволяет сдвинуть элементы.

### 3.3.2 Сортировка вставками с бинарным поиском

Полностью аналогична предыдущей сортировке, за исключением того, что вместо линейного поиска используется бинарный.

## 3.4 Сортировка подсчетом

Приведена реализация для произвольного неотрицательного диапазона значений. Линейным поиском ищется максимум и минимум в диапазоне значений. Создаётся вспомогательный массив C размера max – min + 1, заполненный нулями. Подсчитывается количество вхождений каждого элемента из диапазона. Для каждого элемента j диапазона значений в исходный массив последовательно записать этот самый элемент C[j - min] раз.

## 3.5 Быстрая сортировка

Выбрать из массива элемент, называемый опорным. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на два непрерывных отрезка, следующих друг за другом: элементы «меньшие опорного», «равные и большие». Для этих отрезков выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

## 3.6 Сортировка слиянием

Сортируемый [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) разбивается на две части примерно равного размера. Каждая из них сортируется отдельно тем же алгоритмом. Два упорядоченных массива половинного размера соединяются в один.

## 3.7 Линейный поиск

Линейный поиск последовательно проверяет каждый элемент массива до тех пор, пока не найдет элемент, соответствующий целевому значению.

## 3.8 Бинарный поиск

Бинарный поиск работает только с отсортированными массивами. Бинарный поиск начинается со сравнения элемента в середине массива с целевым значением. Если целевое значение совпадает с элементом, возвращается его позиция в массиве. Если целевое значение меньше элемента, поиск продолжается в половине массива до середины. Если целевое значение больше элемента, поиск продолжается в половине массива после середины. Делая это, алгоритм устраняет половину, в которой целевое значение не может находиться.

# 4.ОПИСАНИЕ ПРОГРАММЫ

## Описание глобальных переменных

int n – размер массивов.

int output\_flag – флаг, отвечающий за вывод массивов в консоль.

int \*arr – эталонный массив.

int \*arr1 – рабочий массив.

## Описание функций

void swap(int\* a, int\* b) – обменивает значения a и b.

void update\_seed\_rand() – обновляет семя рандома.

void copy\_array\_values(int arr[], int arr1[]) – копирует значения массива arr в массив arr1.

void update\_values(int arr[]) – обновляет значения в массиве arr. Если значение глобальной переменной n > 10, то массив будет заполнен случайными значениями. Иначе, будет использован пользовательский ввод.

void update\_size(int\*\* arr, int\*\* arr1) – изменяет размер массивов arr и arr1. В реализации задействует функции update\_values и copy\_array\_values.

void print\_array(int arr[]) – выводит массив arr в консоль.

void print\_search(int index, unsigned long long c\_i, double timer) – обрабатывает и выводит в консоль результат работы алгоритмов поиска.

index – результат работы поиска;

c\_i – счетчик итераций;

timer – время работы поиска (в секундах).

int line\_search(int arr[], int n, int key, unsigned long long\* counter\_iterations, int\* opt = NULL) – реализация алгоритма линейного поиска элемента key в массиве arr.

counter\_iterations – переменная, в которую будет записано значение счётчика итераций.

int binary\_search(int arr[], int n, int key,

unsigned long long\* counter\_iterations, int\* opt = NULL) – реализация алгоритма бинарного поиска элемента key в массиве arr.

counter\_iterations – переменная, в которую будет записано значение счётчика итераций.

opt – если передано значение отличное от NULL, то: если элемент key найден, то в opt будет записан его индекс, иначе в opt будет записан индекс, по которому можно разместить key не нарушая сортировки.

void stupid\_bubble\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) – наивная реализация пузырьковой сортировки массива arr.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps - переменная, в которую будет записано значение счётчика обменов значений.

void bubble\_sort\_with\_break(int arr[],

unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) – реализация пузырьковой сортировки с прерыванием.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void shake\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) – реализация шейкер-сортировки (модификация пузырьковой сортировки)

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void insert\_sort\_with\_line\_search(int arr[],

unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) – реализация сортировки вставками с использованием линейного поиска.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void insert\_sort\_with\_binary\_search(int arr[],

unsigned long long\* counter\_comparisons, unsigned long long\* counter\_swaps) – реализация сортировки вставками с использованием бинарного поиска.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void selection\_sort(int arr[], unsigned long long\* counter\_comparisons, unsigned long long\* counter\_swaps) – реализация сортировки выбором.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void calculation\_sort(int arr[], unsigned long long\* counter\_iterantions) – реализация сортировки подсчётом.

counter\_iterations – переменная, в которую будет записано значение счётчика итераций.

void quick\_sort(int arr[], int first, int last,

unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps,

unsigned long long\* deep\_recursion, int deep = 0) – реализация быстрой сортировка (сортировки Хоара).

first – индекс первого элемента подмассива массива arr;

last – индекс последнего элемента подмассива массива arr;

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений;

deep\_recursion – переменная, в которую будет записано максимальное значение глубины рекурсии;

deep – текущая глубина рекурсии.

void q\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion) – функция-оболочка для функции quick\_sort.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений;

deep\_recursion – переменная, в которую будет записано максимальное значение глубины рекурсии.

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c, unsigned long long\* c\_s)) – функция, вызывающая функции-сортировки.

function\_pointer – указатель на функцию, которая будет вызвана.

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c, unsigned long long\* c\_s, unsigned long long\* deep)) – перегрузка функции call\_sort.

function\_pointer – указатель на функцию, которая будет вызвана.

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_i)) – перегрузка функции call\_sort.

function\_pointer – указатель на функцию, которая будет вызвана.

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

unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) – функция, реализующая слияние массивов.

left – индексy начала первого подмассива;

mid – индекс конца первого подмассива;

right – индекс конца второго подмассива;

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений.

void merge\_sort(int arr[], int left, int right,

unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion,

unsigned long long deep = 0) – функция, реализующая рекурсивную часть сортировки слиянием.

left – индекс левой границы;

right – индекс правой границы;

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений;

deep\_recursion – переменная, в которую будет записано максимальное значение глубины рекурсии;

deep – текущая глубина рекурсии.

void m\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion) – функция-оболочка для функции merge\_sort.

counter\_comparisons – переменная, в которую будет записано значение счётчика сравнений;

counter\_swaps – переменная, в которую будет записано значение счётчика обменов значений;

deep\_recursion – переменная, в которую будет записано максимальное значение глубины рекурсии.

# 5.РЕЗУЛЬТАТЫ

Для массива, содержащего 110 тыс. случайно заданных элементов были получены результаты работы сортировок:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сортировка | Количество сравнений | Количество обменов | Количество итераций | Глубина рекурсии | Время  исполнения, сек |
| Наивная  пузырьковая сортировка | 12099890000 | 3029403442 | – | – | 61.828999 |
| Пузырьковая сортировка с прерыванием | 6049626199 | 3029403442 | – | – | 54.457001 |
| Шейкер-сортировка | 4544037919 | 3029403442 | – | – | 36.471001 |
| Сортировка выбором | 6050055000 | 109989 | – | – | 8.957000 |
| Сортировка вставками с линейным поиском | 3020651549 | 3029403442 | – | – | 18.495001 |
| Сортировка вставками с бинарным поиском | 1456305 | 3029539112 | – | – | 3.684000 |
| Сортировка подсчётом | – | – | 362768 | – | 0.002000 |
| Быстрая сортировка | 1881922 | 520169 | – | 37 | 0.025000 |
| Сортировка слиянием | 1705054 | 3697856 | – | 17 | 0.061000 |

Для отсортированного массива в 100 млн. случайно заданных элементов были получены результаты работы поисков:

|  |  |  |
| --- | --- | --- |
| Поиск | Количество итераций | Время исполнения, сек |
| Линейный поиск | 99706721 | 0.067000 |
| Бинарный поиск | 9 | 0.000000 |

# 6.ЗАКЛЮЧЕНИЕ

Из рассматриваемых сортировок самый лучший результат показала сортировка подсчётом, но из-за ограниченности её применения, лучшей будем считать быструю сортировку.

Из двух рассматриваемых методов поиска, очевидно, лучшим является бинарный поиск.

# 7.ЛИТЕРАТУРА

# 8.ПРИЛОЖЕНИЕ

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

// Global Variable

int n, output\_flag;

// Reference Array

int \*arr;

// Work Array

int \*arr1;

void swap(int\* a, int\* b);

void update\_seed\_rand();

void copy\_array\_values(int arr[], int arr1[]);

void update\_values(int arr[]);

void update\_size(int\*\* arr, int\*\* arr1);

void print\_array(int arr[], int output\_flag);

void print\_search(int index, unsigned long long c\_i, double timer);

int line\_search(int arr[], int n, int key, unsigned long long\* counter\_iterations,

int\* opt = NULL);

int binary\_search(int arr[], int n, int key, unsigned long long\* counter\_iterations,

int\* opt = NULL);

void stupid\_bubble\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void bubble\_sort\_with\_break(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void insert\_sort\_with\_line\_search(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void insert\_sort\_with\_binary\_search(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void selection\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void shake\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps);

void calculation\_sort(int arr[], unsigned long long\* counter\_iterantions);

void quick\_sort(int arr[], int first, int last, unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion, int deep = 0);

void q\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion);

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c,

unsigned long long\* c\_s));

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c,

unsigned long long\* c\_s, unsigned long long\* deep));

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_i));

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

unsigned long long\* counter\_comparisons, unsigned long long\* counter\_swaps);

void merge\_sort(int arr[], int left, int right, unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion,

unsigned long long deep = 0);

void m\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion);

int main() {

int op, op1, loop\_flag, key, response;

unsigned long long c\_c = 0, c\_s = 0, c\_i = 0;

clock\_t start;

do {

printf("Enter lenght array: ");

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

} while (n <= 0);

arr = (int\*)malloc(sizeof(int) \* n);

arr1 = (int\*)malloc(sizeof(int) \* n);

update\_seed\_rand();

update\_values(arr);

copy\_array\_values(arr, arr1);

printf("Print arrays? (1/0): ");

scanf\_s("%d", &output\_flag);

printf("Config is complite.\n\n");

while (1) {

printf("Menu: \n"

"1. Config\n"

"2. Sorts\n"

"3. Searches\n"

"0. Exit\n"

"Enter operation: ");

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

printf("\n");

switch (op)

{

case 0:

system("cls");

printf("See you next time.\n"

"@ Yermolaev Vladislav\n");

return 0;

break;

case 1:

system("cls");

loop\_flag = 1;

while (loop\_flag) {

printf("Config: \n"

"1. Update seed rand\n"

"2. Update values in array\n"

"3. Toggle print arrays\n"

"4. Resize arrays\n"

"5. Print current array\n"

"0. To main menu\n"

"Enter operation: ");

scanf\_s("%d", &op1);

printf("\n");

switch (op1)

{

case 1:

system("cls");

update\_seed\_rand();

printf("Successfully updated.\n\n");

break;

case 2:

system("cls");

update\_values(arr);

printf("Successfully updated.\n\n");

break;

case 3:

system("cls");

printf("Enter (1/0): ");

scanf\_s("%d", &output\_flag);

printf("Successfully updated.\n\n");

break;

case 4:

system("cls");

update\_size(&arr, &arr1);

printf("Successfully updated.\n\n");

break;

case 5:

system("cls");

print\_array(arr, 1);

printf("\n");

break;

case 0:

loop\_flag = 0;

system("cls");

break;

default:

system("cls");

printf("Operation not found.\n\n");

}

}

break;

case 2:

system("cls");

loop\_flag = 1;

while (loop\_flag) {

printf("Sorts: \n"

"1. All sorts\n"

"2. Stupid bubble sort\n"

"3. Bubble sort with break\n"

"4. Shake sort\n"

"5. Select sort\n"

"6. Insert sort with line search\n"

"7. Insert sort with binary search\n"

"8. Calculation sort\n"

"9. Quick sort\n"

"10. Merge sort\n"

"0. To main menu\n"

"Enter operation: ");

scanf\_s("%d", &op1);

printf("\n");

switch (op1)

{

case 1:

system("cls");

case 2:

if (op1 != 1) system("cls");

printf("Stupid bubble sort:\n");

call\_sort(stupid\_bubble\_sort);

if (op1 != 1) break;

case 3:

if (op1 != 1) system("cls");

printf("Bubble sort with break:\n");

call\_sort(bubble\_sort\_with\_break);

if (op1 != 1) break;

case 4:

if (op1 != 1) system("cls");

printf("Shake sort:\n");

call\_sort(shake\_sort);

if (op1 != 1) break;

case 5:

if (op1 != 1) system("cls");

printf("Select sort:\n");

call\_sort(selection\_sort);

if (op1 != 1) break;

case 6:

if (op1 != 1) system("cls");

printf("Insert sort with line search:\n");

call\_sort(insert\_sort\_with\_line\_search);

if (op1 != 1) break;

case 7:

if (op1 != 1) system("cls");

printf("Insert sort with binary search:\n");

call\_sort(insert\_sort\_with\_binary\_search);

if (op1 != 1) break;

case 8:

if (op1 != 1) system("cls");

printf("Calculation sort:\n");

call\_sort(calculation\_sort);

if (op1 != 1) break;

case 9:

if (op1 != 1) system("cls");

printf("Quick sort:\n");

call\_sort(q\_sort);

if (op1 != 1) break;

case 10:

if (op1 != 1) system("cls");

printf("Merge sort:\n");

call\_sort(m\_sort);

break;

case 0:

loop\_flag = 0;

system("cls");

break;

default:

system("cls");

printf("Operation not found\n\n");

}

}

break;

case 3:

system("cls");

loop\_flag = 1;

while (loop\_flag) {

printf("Searches:\n"

"1. All searches\n"

"2. Line search\n"

"3. Binary search\n"

"0. To main menu\n"

"Enter operation: ");

scanf\_s("%d", &op1);

printf("\n");

switch (op1)

{

case 0:

loop\_flag = 0;

system("cls");

break;

case 1:

system("cls");

selection\_sort(arr1, &c\_c, &c\_s);

print\_array(arr1, output\_flag);

printf("Enter digit: ");

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

printf("\n");

case 2:

if (op1 != 1) {

system("cls");

selection\_sort(arr1, &c\_c, &c\_s);

print\_array(arr1, output\_flag);

printf("Enter digit: ");

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

printf("\n");

}

printf("Line search:\n");

start = clock();

response = line\_search(arr1, n, key, &c\_i);

print\_search(response, c\_i, (clock() - start) /

(float)CLOCKS\_PER\_SEC);

if (op1 != 1) {

copy\_array\_values(arr, arr1);

break;

}

case 3:

if (op1 != 1) {

system("cls");

selection\_sort(arr1, &c\_c, &c\_s);

print\_array(arr1, output\_flag);

printf("Enter digit: ");

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

printf("\n");

}

printf("Binary search:\n");

start = clock();

response = binary\_search(arr1, n, key, &c\_i);

print\_search(response, c\_i, (clock() - start) /

(float)CLOCKS\_PER\_SEC);

copy\_array\_values(arr, arr1);

break;

default:

system("cls");

printf("Operation not found\n\n");

}

}

break;

default:

system("cls");

printf("Operation not found\n\n");

}

}

free(arr);

free(arr1);

}

int line\_search(int arr[], int n, int key, unsigned long long\* counter\_iterations, int\* opt) {

int i, c\_i = 0, ind = -1;

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

if (arr[i] == key) {

ind = i;

break;

}

c\_i++;

}

\*counter\_iterations = c\_i;

return ind;

}

int binary\_search(int arr[], int n, int key, unsigned long long\* counter\_iterations, int\* opt)

{

int l = 0, r = n - 1, mid;

unsigned long long c\_i = 0;

while (r >= l) {

mid = (l + r) / 2;

if (arr[mid] == key) {

\*counter\_iterations = c\_i;

if (opt != NULL)

\*opt = mid;

return mid;

}

else if (arr[mid] > key)

r = mid - 1;

else

l = mid + 1;

c\_i++;

}

if (l > r && opt != NULL) {

\*opt = l;

}

\*counter\_iterations = c\_i;

return -1;

}

void swap(int\* a, int\* b) {

int c = \*a;

\*a = \*b;

\*b = c;

}

void update\_seed\_rand() {

srand((unsigned int)time(NULL));

}

void copy\_array\_values(int arr[], int arr1[]) {

int i;

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

arr1[i] = arr[i];

}

}

void update\_values(int arr[]) {

int i;

if (n > 10)

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

arr[i] = rand();

else

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

printf("Enter digit (index %d): ", i);

scanf\_s("%d", &arr[i]);

}

}

void update\_size(int\*\* arr, int\*\* arr1) {

free(\*arr);

free(\*arr1);

do {

printf("Enter new size: ");

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

} while (n <= 0);

\*arr = (int\*)malloc(sizeof(int) \* n);

\*arr1 = (int\*)malloc(sizeof(int) \* n);

update\_values(\*arr);

copy\_array\_values(\*arr, \*arr1);

}

void print\_array(int arr[]) {

int i;

if (output\_flag) {

printf("Current array: ");

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

printf("%d ", arr[i]);

printf("\n");

}

}

void print\_search(int index, unsigned long long c\_i, double timer) {

if (index == -1)

printf("Not found!\n");

else

printf("Element found on index %d\n", index);

printf("Iterations: %lli\n", c\_i);

printf("Time: %lf seconds\n\n", timer);

}

void stupid\_bubble\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j;

unsigned long long c\_s = 0, c\_c = 0;

for (j = 0; j < n; j++) {

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

if (arr[i] > arr[i + 1]) {

swap(&arr[i], &arr[i + 1]);

c\_s++;

}

c\_c++;

}

}

\*counter\_comparisons = c\_c;

\*counter\_swaps = c\_s;

}

void bubble\_sort\_with\_break(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j, flag;

unsigned long long c\_s = 0, c\_c = 0;

for (j = 0; j < n; j++) {

flag = 1;

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

if (arr[i] > arr[i + 1]) {

swap(&arr[i], &arr[i + 1]);

flag = 0;

c\_s++;

}

c\_c++;

}

if (flag)

break;

}

\*counter\_comparisons = c\_c;

\*counter\_swaps = c\_s;

}

void shake\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j, flag;

unsigned long long c\_s = 0, c\_c = 0;

for (j = 0; j < n; j++) {

flag = 1;

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

if (arr[i] > arr[i + 1]) {

swap(&arr[i], &arr[i + 1]);

c\_s++;

flag = 0;

}

c\_c++;

}

if (flag)

break;

for (i = n - j - 1; i > j; i--) {

if (arr[i] < arr[i - 1]) {

swap(&arr[i], &arr[i - 1]);

c\_s++;

flag = 0;

}

c\_c++;

}

if (flag)

break;

}

\*counter\_swaps = c\_s;

\*counter\_comparisons = c\_c;

}

void insert\_sort\_with\_line\_search(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j, ind;

unsigned long long c\_c = 0, c\_s = 0;

for (i = 1; i < n; i++) {

ind = i;

for (j = 0; j < i; j++) {

c\_c++;

if (arr[i] < arr[j]) {

ind = j;

break;

}

}

for (j = i; j > ind; j--) {

swap(&arr[j], &arr[j - 1]);

c\_s++;

}

}

\*counter\_comparisons = c\_c;

\*counter\_swaps = c\_s;

}

void insert\_sort\_with\_binary\_search(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j, ind;

unsigned long long c\_c = 0, c\_s = 0, c\_i;

for (i = 1; i < n; i++) {

binary\_search(arr, i, arr[i], &c\_i, &ind);

for (j = i; j > ind; j--) {

swap(&arr[j], &arr[j - 1]);

c\_s++;

c\_i++;

}

c\_c += c\_i;

}

\*counter\_comparisons = c\_c;

\*counter\_swaps = c\_s;

}

void selection\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps) {

int i, j, mn\_ind;

unsigned long long c\_c = 0, c\_s = 0;

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

mn\_ind = i;

for (j = i; j < n; j++) {

if (arr[j] < arr[mn\_ind]) {

mn\_ind = j;

}

c\_c++;

}

if (mn\_ind != i) {

swap(&arr[mn\_ind], &arr[i]);

c\_s++;

}

}

\*counter\_comparisons = c\_c;

\*counter\_swaps = c\_s;

}

void calculation\_sort(int arr[], unsigned long long\* counter\_iterantions) {

int i, j, k = 0, mn = arr[0], mx = arr[0];

unsigned long long c\_i = 0;

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

if (arr[i] < mn) mn = arr[i];

if (arr[i] > mx) mx = arr[i];

c\_i++;

}

int step = mx - mn + 1;

if (step == 0) step = 1;

int\* counter = (int\*)malloc(sizeof(int) \* step);

if (counter != NULL) {

for (i = 0; i < step; i++) {

counter[i] = 0;

c\_i++;

}

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

counter[arr[i] - mn]++;

c\_i++;

}

for (i = 0; i < step; i++) {

for (j = 0; j < counter[i]; j++) {

arr[k++] = i + mn;

c\_i++;

}

}

}

free(counter);

\*counter\_iterantions = c\_i;

}

void quick\_sort(int arr[], int first, int last, unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion, int deep) {

unsigned long long c\_s = 0, c\_c = 0;

int mid = arr[(first + last) / 2];

int left = first, right = last;

while (left < right) {

while (arr[left] < mid) {

left++;

c\_c++;

}

while (arr[right] > mid) {

right--;

c\_c++;

}

if (left <= right) {

swap(&arr[left], &arr[right]);

c\_c++;

c\_s++;

left++;

right--;

}

}

\*counter\_comparisons += c\_c;

\*counter\_swaps += c\_s;

if (\*deep\_recursion < deep)

\*deep\_recursion = deep;

if (first < right)

quick\_sort(arr, first, right, counter\_comparisons, counter\_swaps,

deep\_recursion, deep + 1);

if (left < last)

quick\_sort(arr, left, last, counter\_comparisons, counter\_swaps,

deep\_recursion, deep + 1);

}

void q\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion) {

quick\_sort(arr, 0, n-1, counter\_comparisons, counter\_swaps, deep\_recursion);

}

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c,

unsigned long long\* c\_s))

{

unsigned long long c\_c = 0, c\_s = 0;

clock\_t start = clock();

int i;

funcion\_pointer(arr1, &c\_c, &c\_s);

if (output\_flag) {

printf("Old array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

printf("Sorted array: ");

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

printf("%d ", arr1[i]);

}

printf("\n");

}

printf("Comparisons: %lli\nSwaps: %lli\n", c\_c, c\_s);

printf("Time: %lf seconds\n", (clock() - start) / (float)CLOCKS\_PER\_SEC);

printf("\n");

copy\_array\_values(arr, arr1);

}

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_c,

unsigned long long\* c\_s, unsigned long long\* deep))

{

unsigned long long c\_c = 0, c\_s = 0, deep = 0;

clock\_t start = clock();

int i;

funcion\_pointer(arr1, &c\_c, &c\_s, &deep);

if (output\_flag) {

printf("Old array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

printf("Sorted array: ");

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

printf("%d ", arr1[i]);

}

printf("\n");

}

printf("Comparisons: %lli\nSwaps: %lli\n", c\_c, c\_s);

printf("Deep recursion: %lli\n", deep);

printf("Time: %lf seconds\n", (clock() - start) / (float)CLOCKS\_PER\_SEC);

printf("\n");

copy\_array\_values(arr, arr1);

}

void call\_sort(void (\*funcion\_pointer) (int arr1[], unsigned long long\* c\_i))

{

unsigned long long c\_i = 0;

clock\_t start = clock();

int i;

funcion\_pointer(arr1, &c\_i);

if (output\_flag) {

printf("Old array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

printf("Sorted array: ");

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

printf("%d ", arr1[i]);

}

printf("\n");

}

printf("Iterations: %lli\n", c\_i);

printf("Time: %lf seconds\n", (clock() - start) / (float)CLOCKS\_PER\_SEC);

printf("\n");

copy\_array\_values(arr, arr1);

}

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

unsigned long long\* counter\_comparisons, unsigned long long\* counter\_swaps) {

int i = left, j = mid + 1, k = 0;

unsigned long long c\_c = 0, c\_s = 0;

int\* tmp = (int\*)malloc(sizeof(int) \* (unsigned long long)(right - left + 1));

if (tmp != NULL) {

while ((i <= mid) && (j <= right)) {

if (arr[i] < arr[j])

tmp[k] = arr[i++];

else

tmp[k] = arr[j++];

k++;

c\_c++;

c\_s++;

}

while (j <= right) {

tmp[k] = arr[j++];

k++;

c\_s++;

}

while (i <= mid) {

tmp[k] = arr[i++];

k++;

c\_s++;

}

for (i = left; i <= right; i++) {

arr[i] = tmp[i - left];

c\_s++;

}

}

\*counter\_comparisons += c\_c;

\*counter\_swaps += c\_s;

free(tmp);

}

void merge\_sort(int arr[], int left, int right, unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion, unsigned long long deep)

{

if (deep > \*deep\_recursion)

\*deep\_recursion = deep;

if (left == right)

return;

int mid = (left + right) / 2;

merge\_sort(arr, left, mid, counter\_comparisons, counter\_swaps, deep\_recursion, deep + 1);

merge\_sort(arr, mid + 1, right, counter\_comparisons, counter\_swaps, deep\_recursion,

deep + 1);

merge\_arrays(arr, left, mid, right, counter\_comparisons, counter\_swaps);

}

void m\_sort(int arr[], unsigned long long\* counter\_comparisons,

unsigned long long\* counter\_swaps, unsigned long long\* deep\_recursion)

{

merge\_sort(arr, 0, n-1, counter\_comparisons, counter\_swaps, deep\_recursion);

}