Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление арифметических выражений»**

**Выполнил**:

студент группы 382003-1

Смолин К.А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Написать программу для сортировки массива чисел типа **double** различными алгоритмами: **сортировкой вставками**, **сортировкой расчёской**, **сортировкой слиянием** и **поразрядной сортировкой**. Для каждой сортировки необходимо посчитать количество перестановок и сравнений. Проверить теоретическую сложность алгоритма.

# Метод решения

**Сортировка вставками (Insertion Sort)** — это простой алгоритм сортировки. На каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Изначально отсортированная часть состоит из одного первого элемента, затем последовательно считываются другие элементы массива и размещаются на подходящее место в уже отсортированной части массива. Массив из 1-го элемента считается отсортированным.

**Сортировка расчёской (Comb Sort)** — алгоритм сортировки массива, является улучшенным вариантом сортировки пузырьком. Основная идея заключается в устранении черепах, или маленьких значений в конце списка, которые крайне замедляют сортировку пузырьком. При этом первоначально необходимо брать большое расстояние, и постепенно уменьшать его, по мере упорядочивания данных вплоть до единицы. Изначально сравнивается первый и последний элемент массива, и на каждой итерации уменьшается разрыв между элементами на фактор уменьшения.

* Оптимальное значение фактора уменьшения равно *1/(1-e-φ) ≈ 1.2473309*, где *е* – основание натурального логарифма, а *φ* – золотое сечение.

Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные подмножества. Элементы двигаются сразу на несколько позиций в сторону своего истинного места.

**Сортировка слиянием (Merge Sort)** — Работает по принципу «Разделяй и Властвуй»

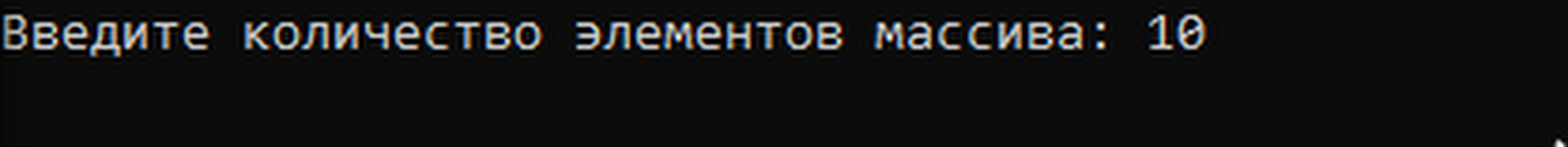
Основная идея состоит в слиянии двух упорядоченных массивов в один. Массив рекурсивно разбивается пополам, и каждая из половин делиться до тех пор, пока размер очередного подмассива не станет равным единице. Далее выполняется операция алгоритма, называемая слиянием. Два единичных массива сливаются в общий результирующий массив, при этом из каждого выбирается меньший элемент (сортировка по возрастанию) и записывается в свободную левую ячейку результирующего массива. После чего из двух результирующих массивов собирается третий общий отсортированный массив, и так далее. В конце операции слияния, элементы перезаписываются из результирующего массива в исходный.

**Поразрядная сортировка (Radix Sort)** — Используется сортировка подсчетом и ее свойство устойчивости. Идея состоит в сортировки ключей по отдельным разрядам, что приведет к упорядочиванию всего массива (Предполагается, что каждый ключ сортировки можно рассматривать как k-значное число, каждая цифра которого находится в диапазоне от 0 до m-1). Поочередно используется устойчивая сортировка (например, сортировка подсчётом) для каждой цифры справа налево. Порядок сортировки цифр или символов важен.

* O(k \* (n+m)) — по времени работы
* O(n+m) — по дополнительной памяти

# Руководство пользователя

При запуске программы пользователю необходимо ввести количество элементов массива.



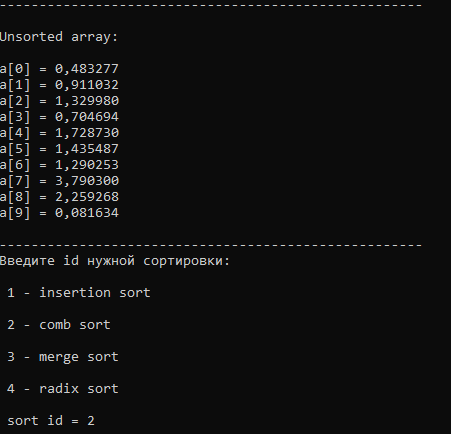
Далее предлагается выбрать тип ввода массива:

1. Random
2. Ручной ввод

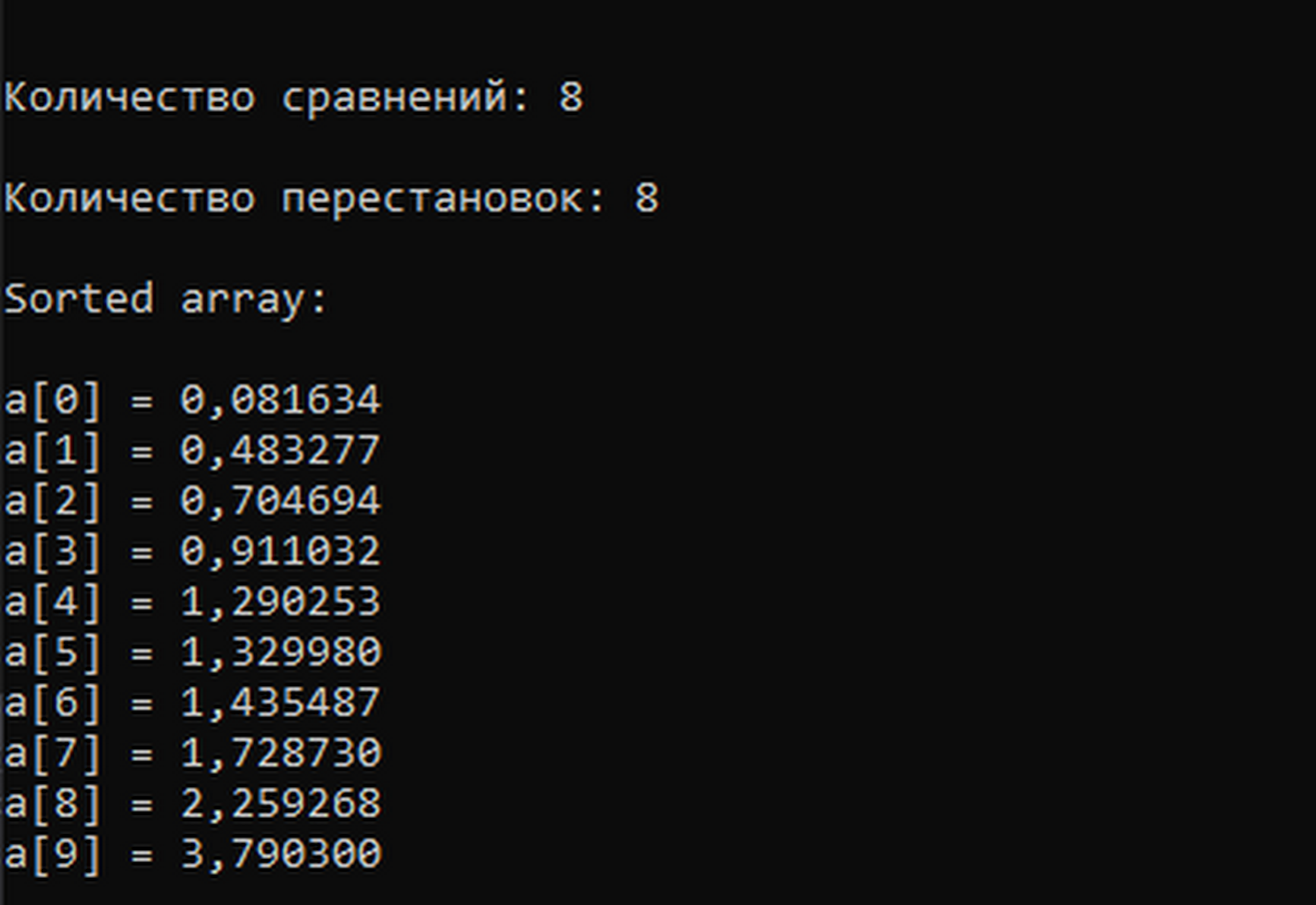


После ввода длины массива и типа ввода, пользователь увидит неотсортированный массив и ему будет предложено выбрать тип сортировки:

1. Сортировка вставками
2. Сортировка расчёской
3. Сортировка слиянием
4. Поразрядная сортировка

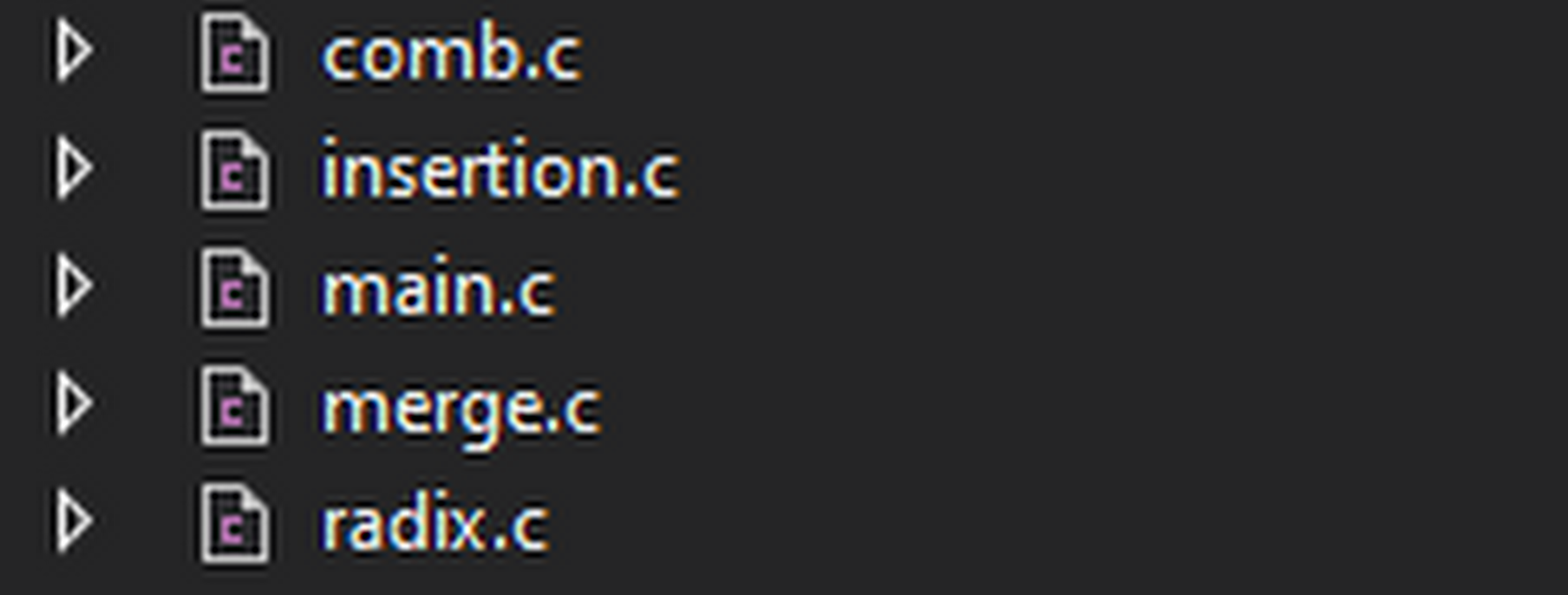


После выбора типа сортировки пользователь получит отсортированный массив, а также увидит количество перестановок и сравнений.



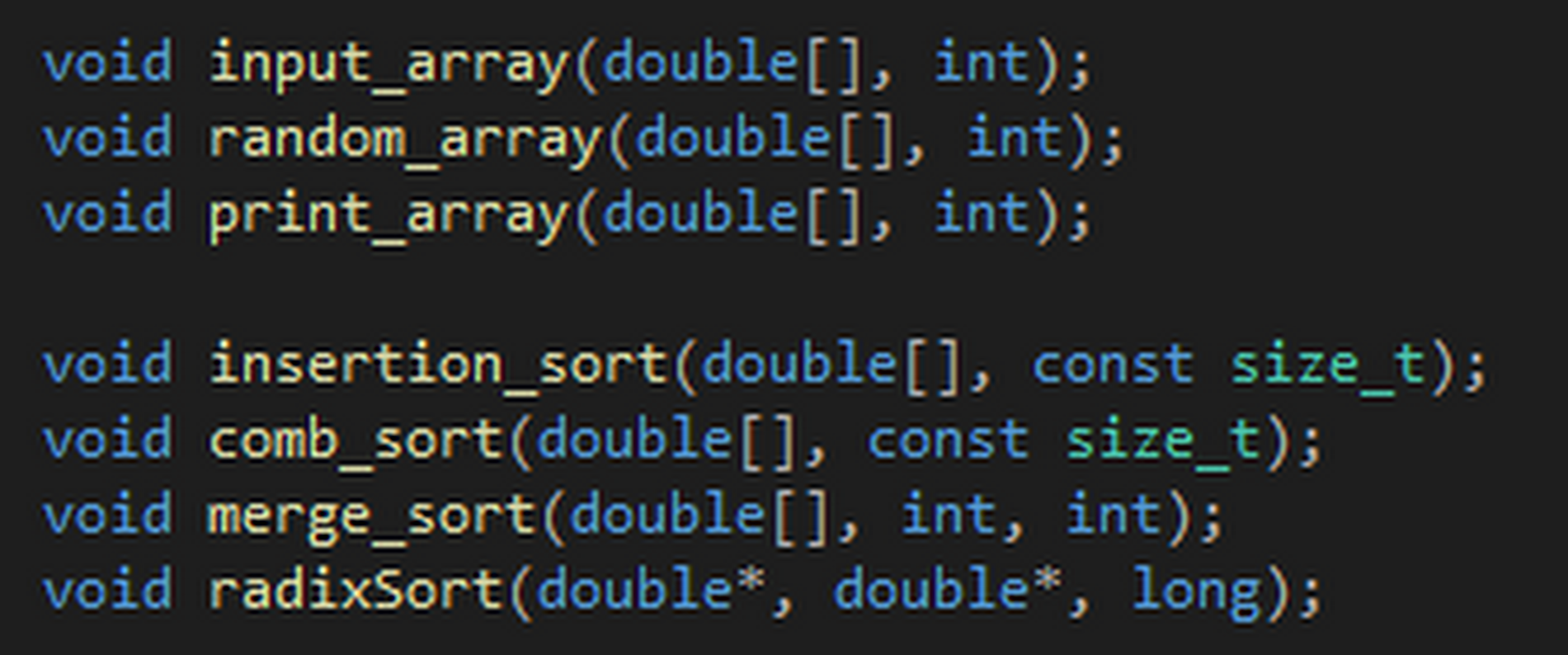
# Описание программной реализации

В проекте используется 5 исходных файлов(.c)



1. Main.c

Главный файл, в котором реализованы функции ввода-вывода массива, реализовано меню для работы с пользователем.



1. Insertion.c

Файл, в котором содержится функция сортировки вставками

1. Comb.c

Файл, в котором содержится функция сортировки расчёской

1. Merge.c

Файл, в котором содержится реализация сортировки слиянием, состоит из 2 функций

1. Radix.c

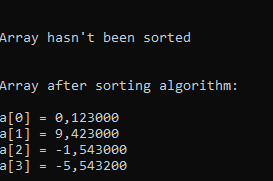
Файл, в котором содержится реализация поразрядной сортировки, состоит из 3 функций

# Подтверждение корректности

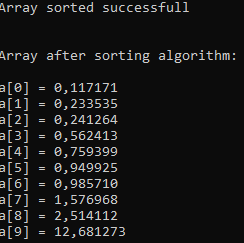
Для подтверждения корректности в программе существует функция, реализованная в main.c: **void correct\_check(double[], int);**

Данная функция выполняется после сортировки и проверяет каждые два последовательных элемента.

Пример проверки неправильной сортировки:



Пример правильной сортировки:



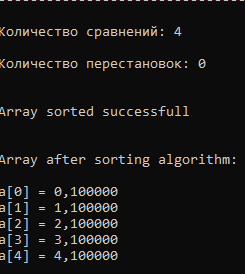
# Результаты экспериментов

**Сортировка вставками**

Сложность O(n^2)

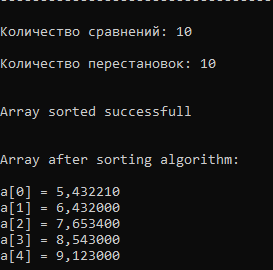
Лучший случай: полностью упорядоченный массив

* Количество сравнений = N
* Количество перестановок = 0



Худший случай: обратно упорядоченный массив

* Количество сравнений = n\*(n-1)/2
* Количество перестановок = n\*(n-1)/2



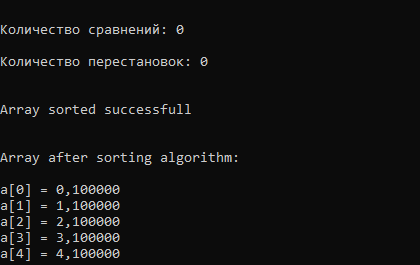
**Сортировка расчёской**

В лучшем случае и среднем случае:

В худшем случае:

**Лучший случай:** полностью упорядоченный массив

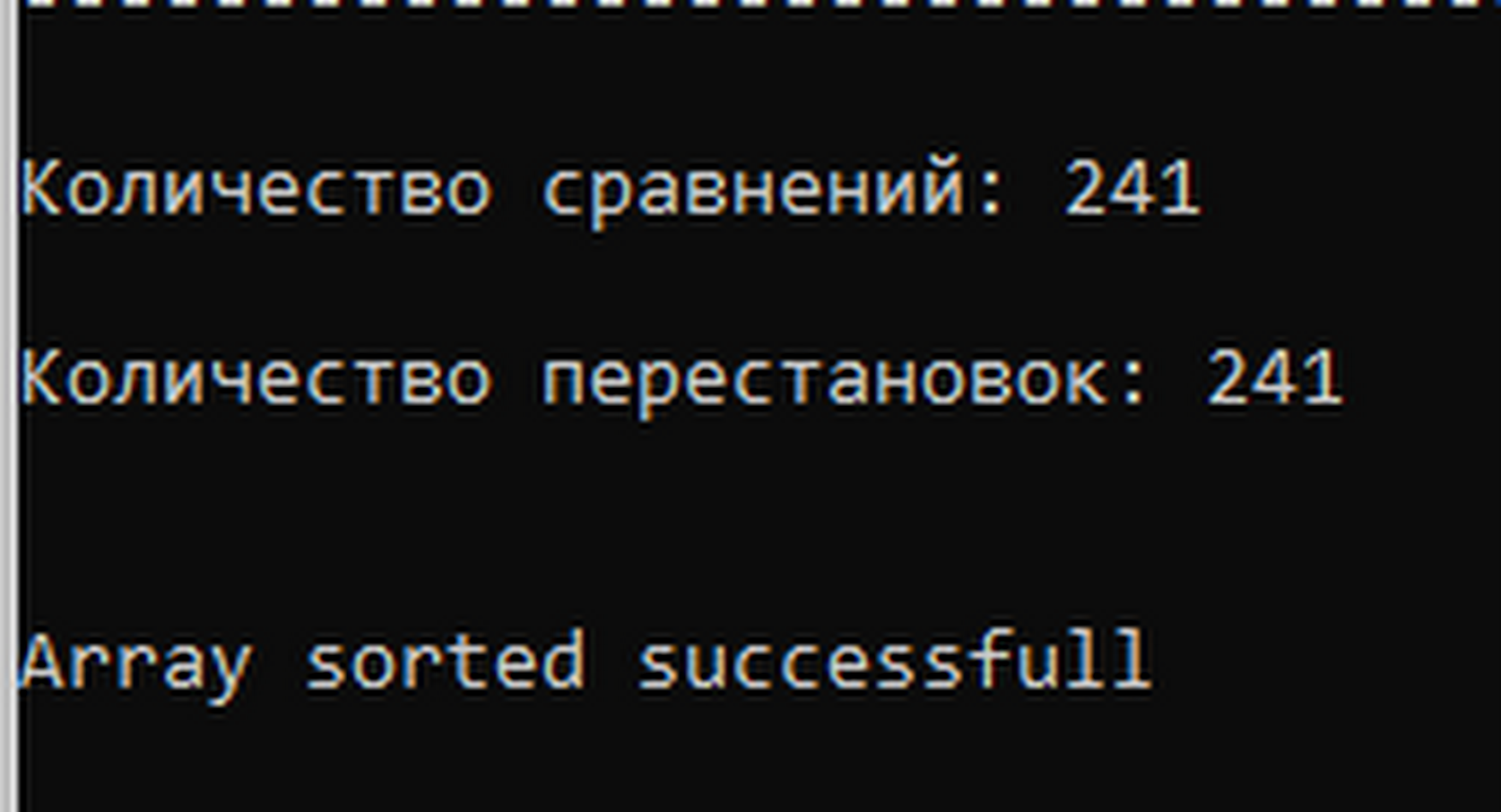
* Количество сравнений = 0
* Количество перестановок = 0



**Худшим случаем** будет такой, при котором на каждом этапе массив будет разделяться на вырожденный подмассив из одного опорного элемента и на подмассив из всех остальных элементов. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых.

**Средний случай:** случайное задание массива(N = 100)

* Количество сравнений ~ n\*log(n)
* Количество перестановок ~ n\*log(n)



**Сортировка слиянием**

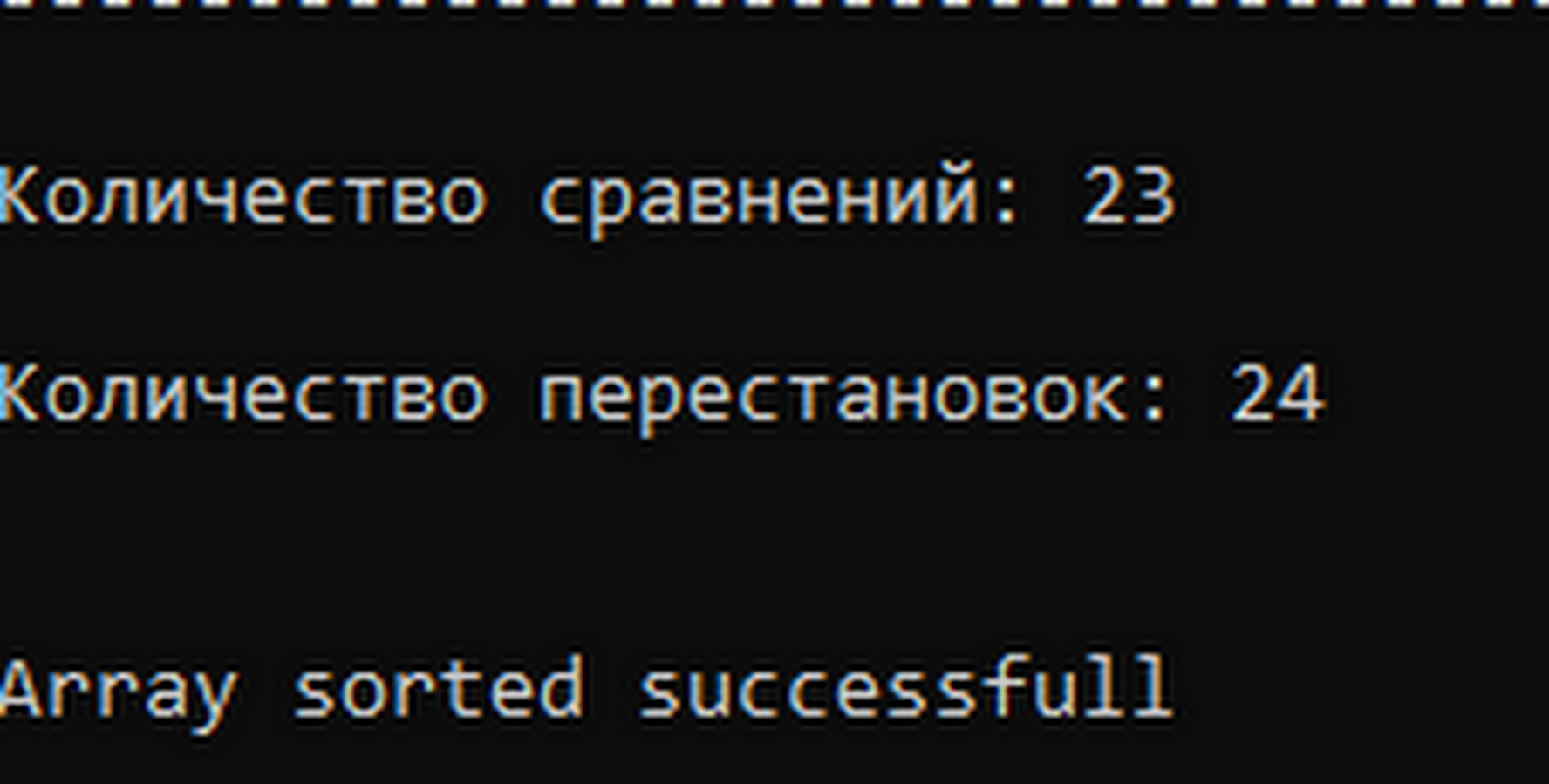
**Сложность алгоритма**: О(n\*log(n))

**Лучший случай**: отсортированный массив

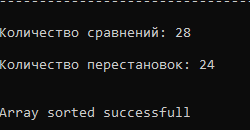
**средним случаем**: случайно заполненный массив.

**Худший случай:** не рассматривается

1. **Отсортированный N = 5**

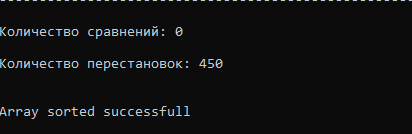


1. **Случайно заполненный N = 5**



**Поразрядная сортировка**

**Сложность**: О(n) **во всех случаях**



# Заключение

В заключение стоит отметить, что поставленная задача была выполнена. Была написана программа для сортировки массива разными алгоритмами, описан принцип работы пользователя с программой, описана структура проекта, способ проверки корректности алгоритмов, приведены результаты экспериментов.

# Приложение

void insertion\_sort(double array[], size\_t size)

{

int k = 0;

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

{

k = i;

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

{

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

{

double tmp = array[j - 1];

array[j - 1] = array[j];

array[j] = tmp;

perm\_insertion++;

comp\_insertion++;

k = j - 1;

}

}

if (k != 0) comp\_insertion++;

}

printf("\nКоличество сравнений: %d\n", comp\_insertion);

printf("\nКоличество перестановок: %d\n", perm\_insertion);

}

void comb\_sort(double array[], const size\_t size) {

double factor = 1.2473309; int step = size - 1;

while (step >= 1) {

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

if (array[i] > array[i + step]) {

double tmp = array[i];

array[i] = array[i + step];

array[i + step] = tmp;

perm\_comb++;

comp\_comb++;

}

}

step /= factor;

}

printf("\nКоличество сравнений: %d\n", comp\_comb);

printf("\nКоличество перестановок: %d\n", perm\_comb);

}

void merge\_sort(double array[], int first, int last) {

int middle = (first + last) / 2;

if (first < last) {

merge\_sort(array, first, middle);

merge\_sort(array, middle+1, last);

merge(array, first, last);

comp\_merge++;

}

}

void merge(double array[], int first, int last) {

int middle, start, final, j;

double\* second;

second = (double\*)malloc((2 \* last) \* sizeof(double));

middle = (first + last) / 2;

start = first;

final = middle + 1;

for (j = first; j <= last; j++) {

if (start <= middle) {

if ((final > last) || (array[start] < array[final]))

{

second[j] = array[start];

start++;

comp\_merge++;

perm\_merge++;

}

else

{

second[j] = array[final];

final++;

comp\_merge++;

perm\_merge++;

}

comp\_merge++;

}

else {

second[j] = array[final];

final++;

comp\_merge++;

perm\_merge++;

}

}

for (j = first; j <= last; j++) {

array[j] = second[j];

perm\_merge++;

}

free(second);

}

void createCounters(double\* data, long\* counters, long N)

{

unsigned char\* bp = (unsigned char\*)data;

unsigned char\* dataEnd = (unsigned char\*)(data + N);

unsigned short i;

memset(counters, 0, 256 \* sizeof(double) \* sizeof(long));

while (bp != dataEnd) {

for (i = 0; i < sizeof(double); i++)

counters[256 \* i + \*(bp++)]++;

}

}

void radixPass(short offset, long N, double\* source, double\* dest, long\* count)

{

double\* sp;

long s = 0, c, i, \* cp = count;

unsigned char\* bp;

for (i = 256; i > 0; --i, cp++) {

c = \*cp; \*cp = s; s += c;

}

bp = (unsigned char\*)source + offset;

sp = source;

for (i = N; i > 0; --i, bp += sizeof(double), ++sp)

{

cp = count + \*bp;

dest[\*cp] = \*sp;

(\*cp)++;

perm\_radix++;

}

}

void radixSort(double\* in, double\* out, long N)

{

long\* count;

long\* counters = (long\*)malloc(sizeof(double) \* 256 \* sizeof(long));

createCounters(in, counters, N);

for (unsigned short i = 0; i < sizeof(double); i++) {

count = counters + 256 \* i;

radixPass(i, N, in, out, count);

for (long j = 0; j < N; j++)

in[j] = out[j];

}

while (in[k] >= 0 && k < N && !(k > 0 && in[k] <= 0 && in[k - 1] > 0)) k++;

for (int i = 0; i < N - k; i++)

in[i] = out[N - 1 - i];

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

in[N - k + i] = out[i];

perm\_radix += N;

printf("\nКоличество сравнений: %d\n", comp\_radix);

printf("\nКоличество перестановок: %d\n", perm\_radix);

free(counters);

}