



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н. Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н. Э. Баумана)

---

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

## ОТЧЕТ

По лабораторной работе №3

По курсу: «Анализ алгоритмов»

Тема: «Алгоритмы сортировки»

Студент:

Пронин А. С.

Группа:

ИУ7-52Б

Преподаватель:

Волкова Л. Л.

Оценка:

\_\_\_\_\_

Москва

2021

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
1.1 Сортировка пузырьком . . . . .	4
1.2 Сортировка выбором . . . . .	4
1.3 Сортировка вставками . . . . .	5
1.4 Вывод . . . . .	5
<b>2 Конструкторский раздел</b>	<b>6</b>
2.1 Требования к ПО . . . . .	6
2.2 Схемы алгоритмов . . . . .	6
2.3 Вывод . . . . .	10
<b>3 Технологический раздел</b>	<b>11</b>
3.1 Выбор инструментов . . . . .	11
3.2 Реализация алгоритмов . . . . .	11
3.3 Тестирование . . . . .	12
3.4 Вывод . . . . .	13
<b>4 Исследовательский раздел</b>	<b>14</b>
4.1 Примеры работы программы . . . . .	14
4.2 Сравнительный анализ времени выполнения алгоритмов .	15
4.3 Оценка трудоёмкости . . . . .	22
4.4 Вывод . . . . .	24
<b>Заключение</b>	<b>25</b>
<b>Список использованных источников</b>	<b>26</b>

# Введение

**Цель работы** – изучение алгоритмов сортировки и получение навыков оценки трудоемкости алгоритмов.

**Задачи работы:**

- реализовать 3 выбранных алгоритма сортировки;
- оценить время выполнения алгоритмов сортировки;
- рассчитать трудоемкость каждого из алгоритма сортировки.

# 1 Аналитический раздел

*Сортировкой* (англ. *sorting*) называется процесс упорядочивания множества объектов по какому-либо признаку.

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов в списке.

Существует огромное количество разнообразных алгоритмов сортировки. Они все отличаются трудоемкостью, скоростью работы.

В данной лабораторной работе были выбраны следующие алгоритмы сортировки:

- сортировка пузырьком;
- сортировка выбором;
- сортировка вставками.

## 1.1 Сортировка пузырьком

Данный алгоритм проходит по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делается так, пока массив не будет отсортирован. Важно отметить, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее. Очевидно, не более чем после  $n$  итераций массив будет отсортирован. Таким образом, асимптотика в худшем и среднем случае —  $O(n^2)$ , в лучшем случае —  $O(n)$ . [1]

## 1.2 Сортировка выбором

На очередной итерации алгоритма находится минимум в массиве после текущего элемента и меняется с ним, если надо. Таким образом, после  $i$ -ой итерации первые  $i$  элементов будут стоять на своих местах. Асимптотика:  $O(n^2)$  в лучшем, среднем и худшем случае. [1]

## 1.3 Сортировка вставками

Создаётся массив, в котором после завершения алгоритма будет находиться ответ. Поочередно вставляются элементы из исходного массива так, чтобы элементы в массиве-ответе всегда были отсортированы. Асимптотика в среднем и худшем случае –  $O(n^2)$ , в лучшем –  $O(n)$ . Реализовывать алгоритм удобнее по-другому (создавать новый массив и реально что-то вставлять в него относительно сложно): сортируется некоторый префикс исходного массива, а вместо вставки меняется текущий элемент с предыдущим, пока они стоят в неправильном порядке. [1]

## 1.4 Вывод

В данном разделе были рассмотрены алгоритмы сортировки пузырьком, выбором и вставками.

## 2 Конструкторский раздел

В данном разделе представлены требования к разрабатываемому ПО и схемы выбранных алгоритмов сортировок.

### 2.1 Требования к ПО

Выбранные алгоритмы сортировки, должны получать на вход целочисленный массив из  $N$  (где  $N \in [0 : 5000]$ ) элементов и возвращать отсортированный по возрастанию массив по месту.

### 2.2 Схемы алгоритмов

Ниже представлены схемы следующих алгоритмов сортировки:

- сортировка пузырьком (рисунок 2.1);
- соритровка выбором (рисунок 2.2);
- сортировка вставками (рисунок 2.3).

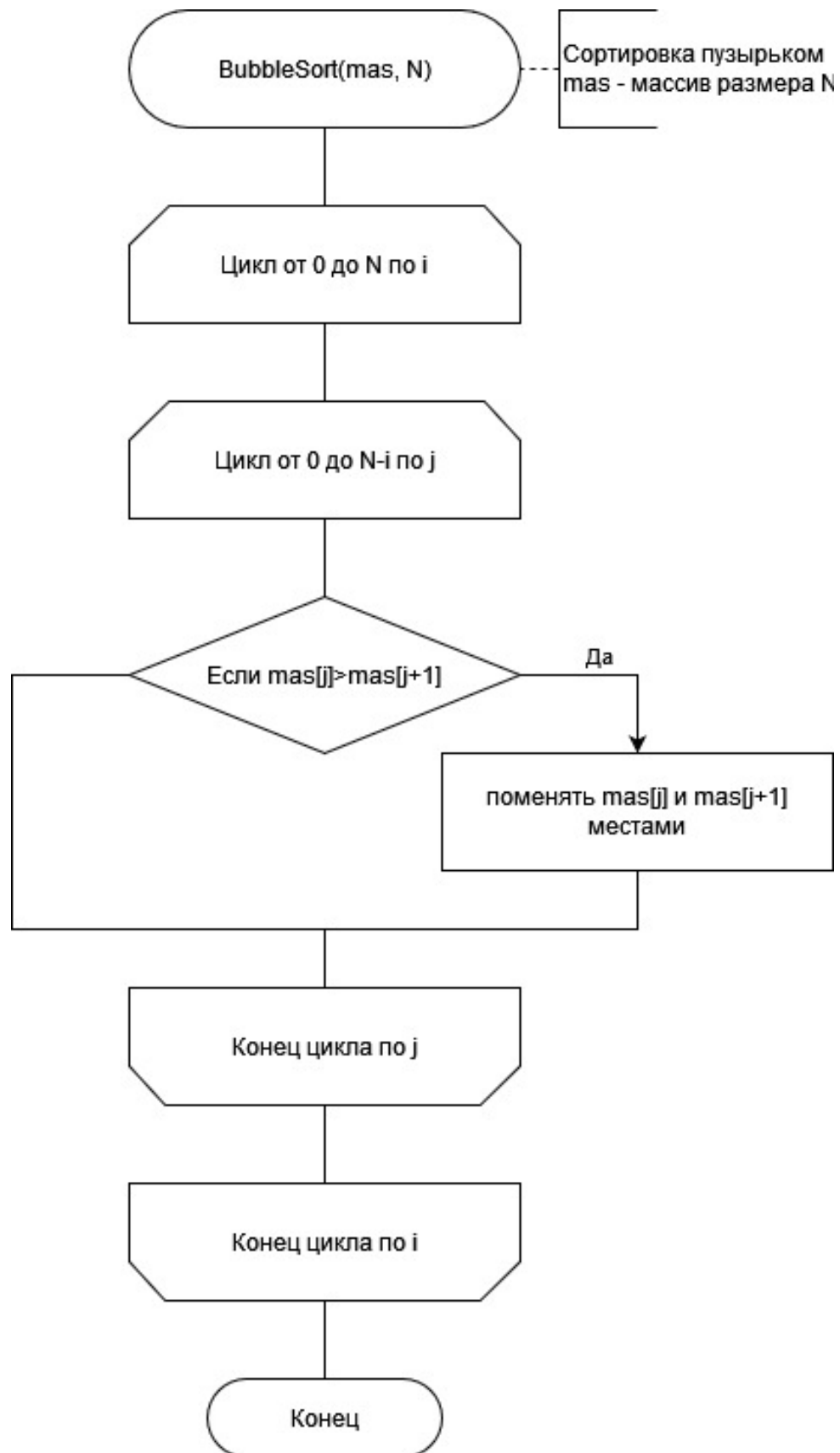


Рис. 2.1: Сортировка пузырьком

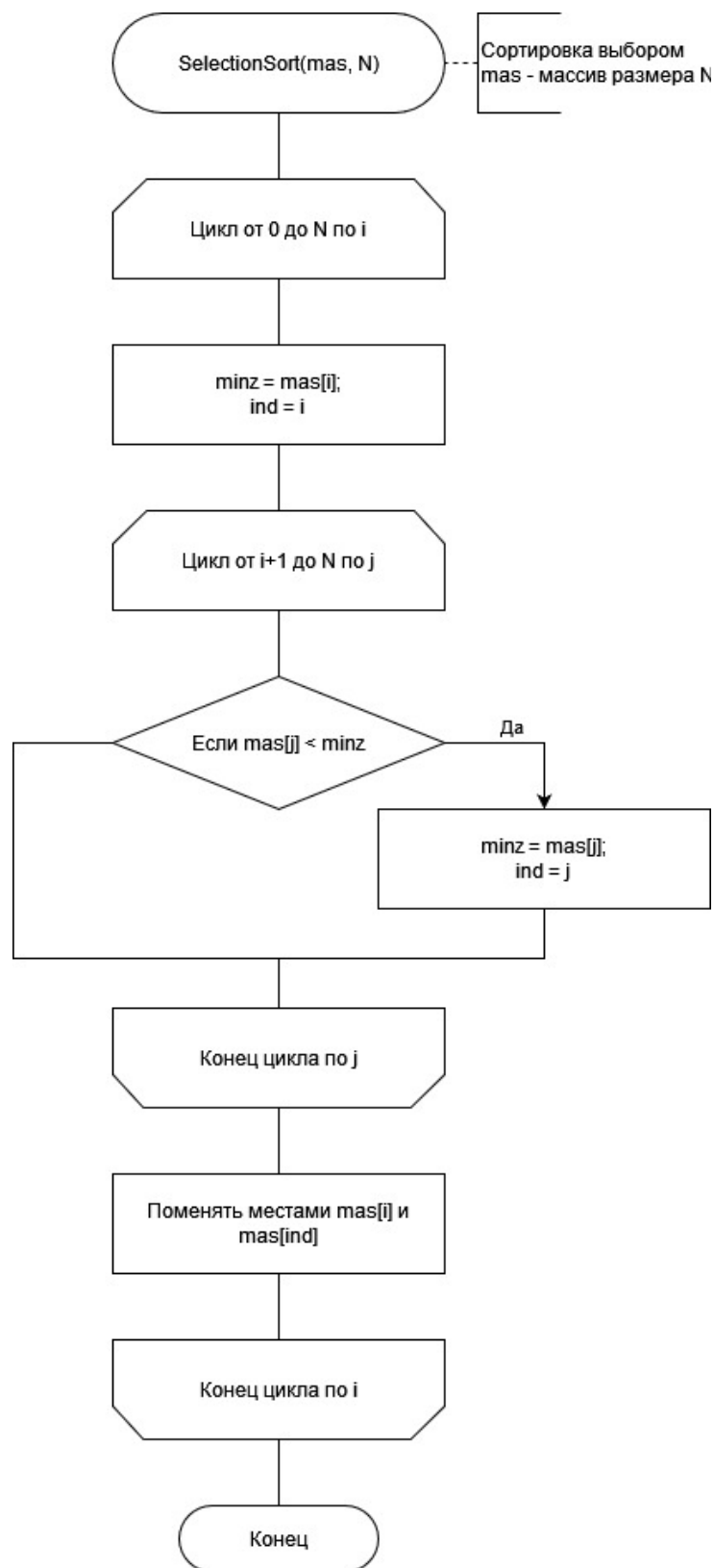


Рис. 2.2: Сортировка выбором



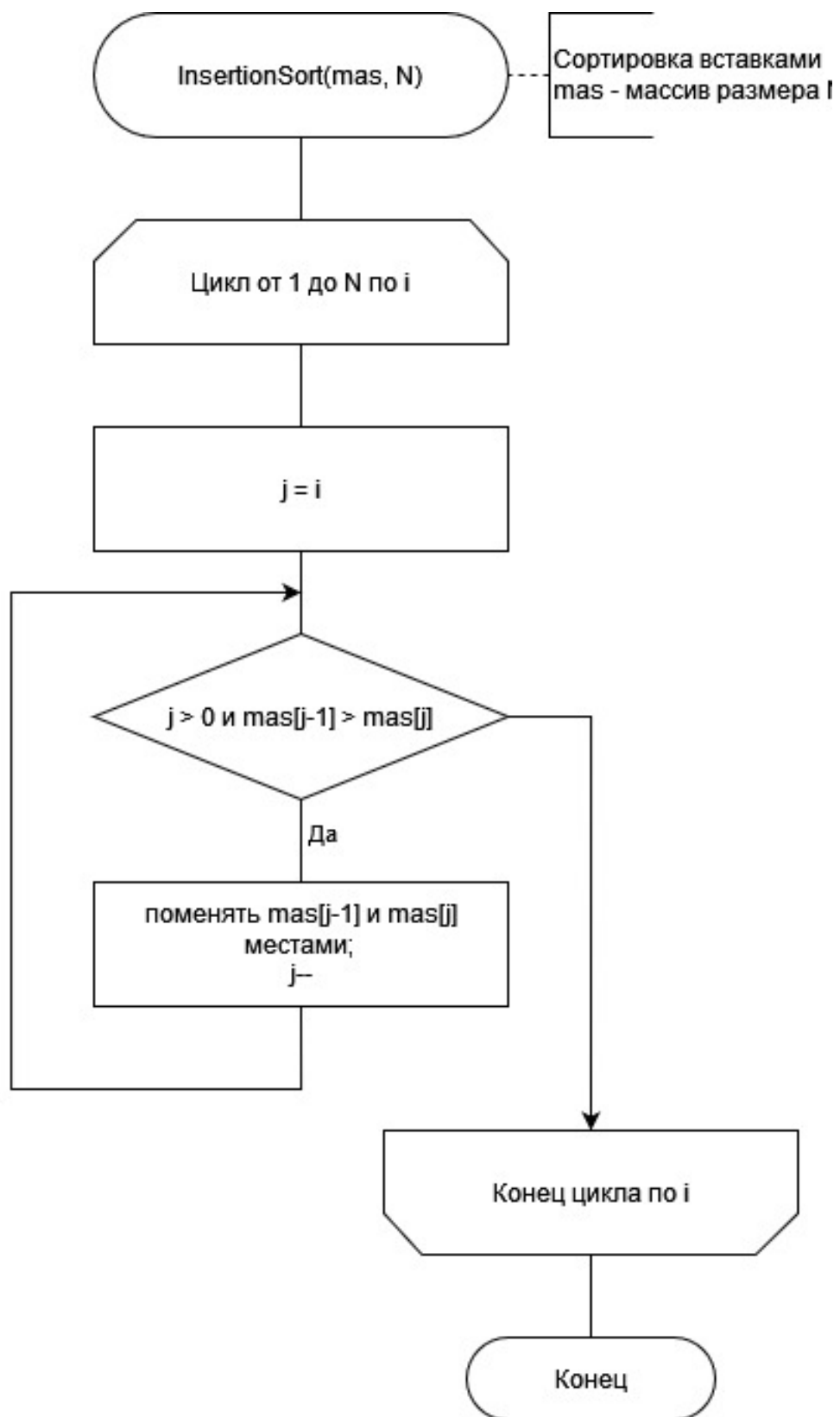


Рис. 2.3: Сортировка вставками

## 2.3 Вывод

В данном разделе были представлены требования к разрабатываемому ПО и разработаны схемы алгоритмов для выбранных сортировок.

## 3 Технологический раздел

В данном разделе представлены выбор инструментов для реализации и оценки алгоритмов, а также листинги полученного кода.

### 3.1 Выбор инструментов

По-скольку наиболее освоенным языком для разработчика является `c++`, для реализации алгоритмов был выбран именно он, т.к. таким образом работа будет проделана наиболее быстро и качественно.

Чтобы оценить время выполнения программы будет замеряться процессорное время, т.к. таким образом будут получены данные подходящие для целесообразного сравнения алгоритмов. Для замера процессорного времени программы используется функция `GetProcessTimes()` т.к. программа пишется под Windows. [2]

Кроме этого, необходимо отключить оптимизации компилятора для более честного сравнения алгоритмов. В моём случае это делается с помощью ключа `-O0` т.к. используется компилятор `G++`. [3]

### 3.2 Реализация алгоритмов

На листингах 3.1-3.3 представлены реализации алгоритмов сортировок пузырьком, выбором и вставками.

Листинг 3.1: Реализация алгоритма сортировки пузырьком

```
1 void BubbleSort(int *l, int *r)
2 {
3     for (int i = 0; i < r-l; i++)
4         for (int *j = l; j < r-i; j++)
5             if (*j > *(j+1))
6                 swap(j, (j+1));
7 }
```

### Листинг 3.2: Реализация алгоритма сортировки выбором

```
1 void SelectionSort(int *l, int *r)
2 {
3     for (int *i = l; i <= r; i++)
4     {
5         int minz = *i, *ind = i;
6         for (int *j = i + 1; j <= r; j++)
7         {
8             if (*j < minz)
9             {
10                 minz = *j;
11                 ind = j;
12             }
13         }
14         swap(i, ind);
15     }
16 }
```

### Листинг 3.3: Реализация алгоритма сортировки вставками

```
1 void InsertionSort(int* l, int* r)
2 {
3     for (int *i = l + 1; i <= r; i++)
4     {
5         int* j = i;
6         while (j > l && *(j - 1) > *j)
7         {
8             swap((j - 1), j);
9             j--;
10        }
11    }
12 }
```

## 3.3 Тестирование

Для проверки написанных алгоритмов были подготовлены функциональные тесты с двумя массивами:

- mas0 = пустой массив
- mas1 = {1};
- mas10 = {10,7,9,5,3,1,8,2,4,6}.

Правильными результатами тестов для них будут следующие массивы:

- `sorted0` = пустой массив
- `sorted1` =  $\{1\}$ ;
- `sorted10` =  $\{1,2,3,4,5,6,7,8,9,10\}$ .

На рисунке 3.1 приведены результаты тестирования.

```
mas0 =
sorted0 by BubbleSort =
BbubbleSort() with mas0 is correct!
sorted0 by BubbleSort =
SelectionSort() with mas0 is correct!
sorted0 by BubbleSort =
InsertionSort() with mas0 is correct!
mas1 = 10
sorted1 by BubbleSort = 10
BbubbleSort() with mas1 is correct!
sorted1 by SelectionSort = 10
SelectionSort() with mas1 is correct!
sorted1 by InsertionSort = 10
SelectionSort() with mas1 is correct!
mas10 = 10 7 9 5 3 1 8 2 4 6
sorted10 by BubbleSort = 1 2 3 4 5 6 7 8 9 10
BbubbleSort() with mas10 is correct!
sorted10 by SelectionSort = 1 2 3 4 5 6 7 8 9 10
SelectionSort() with mas10 is correct!
sorted10 by InsertionSort = 1 2 3 4 5 6 7 8 9 10
InsertionSort() with mas10 is correct!
```

Рис. 3.1: Результаты функционального тестирования

Как видно по рисунку, функциональные тесты пройдены.

## 3.4 Вывод

В данном разделе были выбраны инструменты для реализации выбранных алгоритмов, представлены листинги данных сортировок, а также проведено функциональное тестирование.

## 4 Исследовательский раздел

В данном разделе представлены примеры работы программы, сравнительный анализ реализованных алгоритмов и оценка их трудоёмкости.

### 4.1 Примеры работы программы

На рисунках 4.1-4.3 представлены результаты работы программы для массивов разных длин, заполненных случайными значениями.

```
Array:
46 22
BubbleSort:
22 46
SelectionSort:
22 46
InsertionSort:
22 46
```

Рис. 4.1: Результаты сортировки массива с размером  $= 2$

```
Array:
59 34 92 33 4 22 49 38 60 99
BubbleSort:
4 22 33 34 38 49 59 60 92 99
SelectionSort:
4 22 33 34 38 49 59 60 92 99
InsertionSort:
4 22 33 34 38 49 59 60 92 99
```

Рис. 4.2: Результаты сортировки массива с размером  $= 10$

```

Array:
55 31 91 52 25 47 63 3 37 0 41 75 69 71 27 93 96 6 52 85 32 51 5 79 95 13 60 83 98 97
BubbleSort:
0 3 5 6 13 25 27 31 32 37 41 47 51 52 52 55 60 63 69 71 75 79 83 85 91 93 95 96 97 98
SelectionSort:
0 3 5 6 13 25 27 31 32 37 41 47 51 52 52 55 60 63 69 71 75 79 83 85 91 93 95 96 97 98
InsertionSort:
0 3 5 6 13 25 27 31 32 37 41 47 51 52 52 55 60 63 69 71 75 79 83 85 91 93 95 96 97 98

```

Рис. 4.3: Результаты сортировки массива с размером = 30

## 4.2 Сравнительный анализ времени выполнения алгоритмов

Чтобы провести сравнительный анализ времени выполнения алгоритмов замерялось процессорное время для массивов с 100, 200, ... 1000 элементами. Чтобы оценить время выполнения сортировки для массива размера  $N$ , он заполнялся числами от 0  $N^{10} - 1$ , замерялось процессорное время для части кода, которая сортировала массивы  $500000/N$  раз, после чего результат делился на кол-во итераций.

Сравнительный анализ проводилось на компьютере с процессором AMD Ryzen 5 5600H.

Для сортировки пузырьком наихудшим случаем является массив отсортированный в обратном порядке. Наилучшим случаем является полностью отсортированный массив. На рисунке 4.4 изображены зависимости времени выполнения сортировки от длины массива для произвольного, лучшего и худшего случаев. [1]

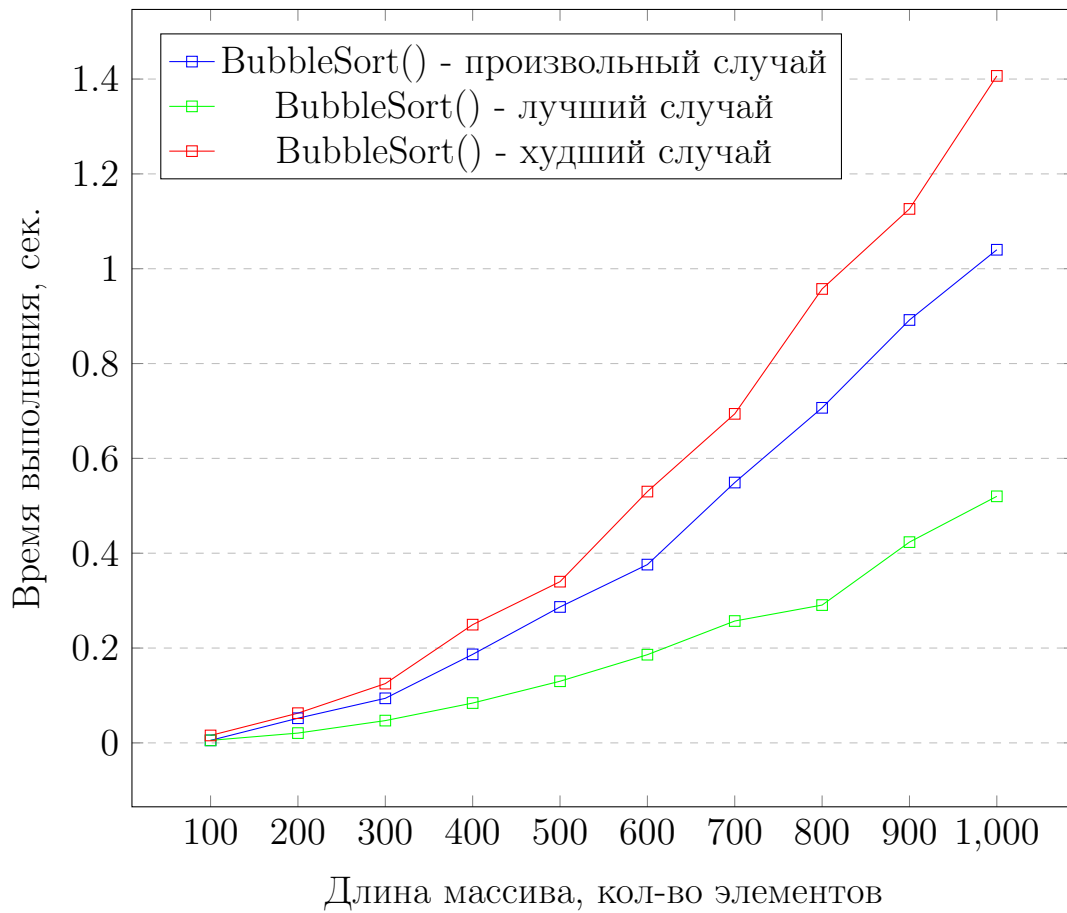


Рис. 4.4: Зависимость времени выполнения сортировки пузырьком от длины массива в разных случаях



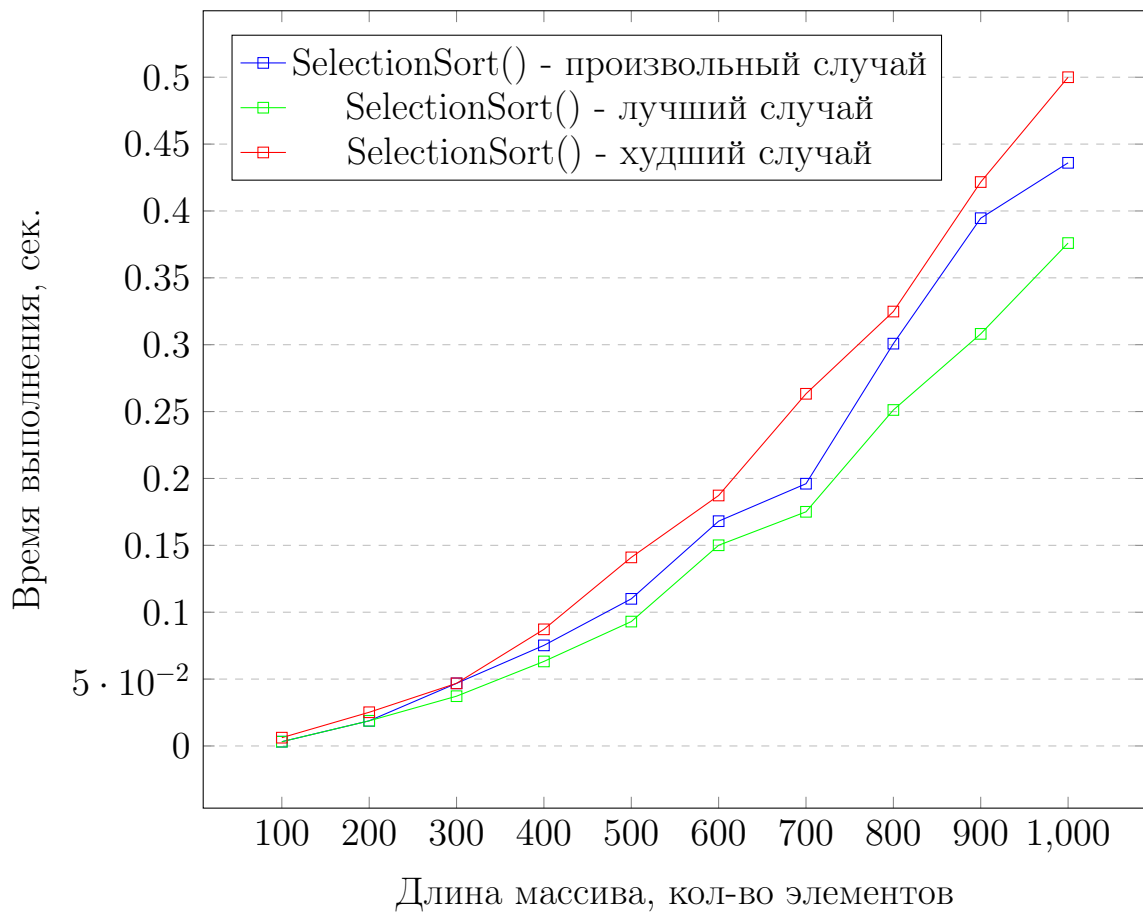


Рис. 4.5: Зависимость времени выполнения сортировки выбором от длины массива в разных случаях

Для сортировки выбором наихудшим случаем является массив отсортированный в обратном порядке. Наилучшим случаем является полностью отсортированный массив. На рисунке 4.5 изображены зависимости времени выполнения сортировки от длины массива для произвольного, лучшего и худшего случаев. [1]

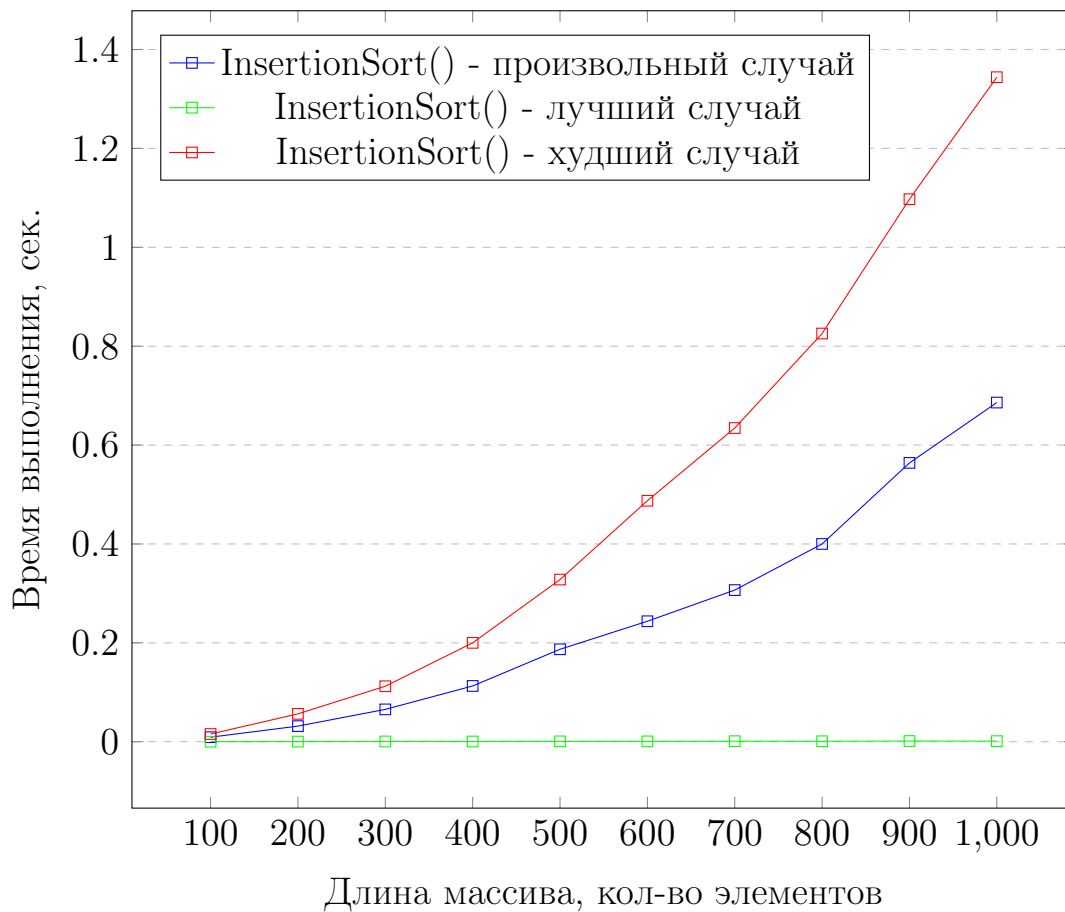


Рис. 4.6: Зависимость времени выполнения сортировки выбором от длины массива в разных случаях

Для сортировки вставками наихудшим случаем является массив отсортированный в обратном порядке. Наилучшим случаем является полностью отсортированный массив. На рисунке 4.6 изображены зависимости времени выполнения сортировки от длины массива для произвольного, лучшего и худшего случаев. [1]

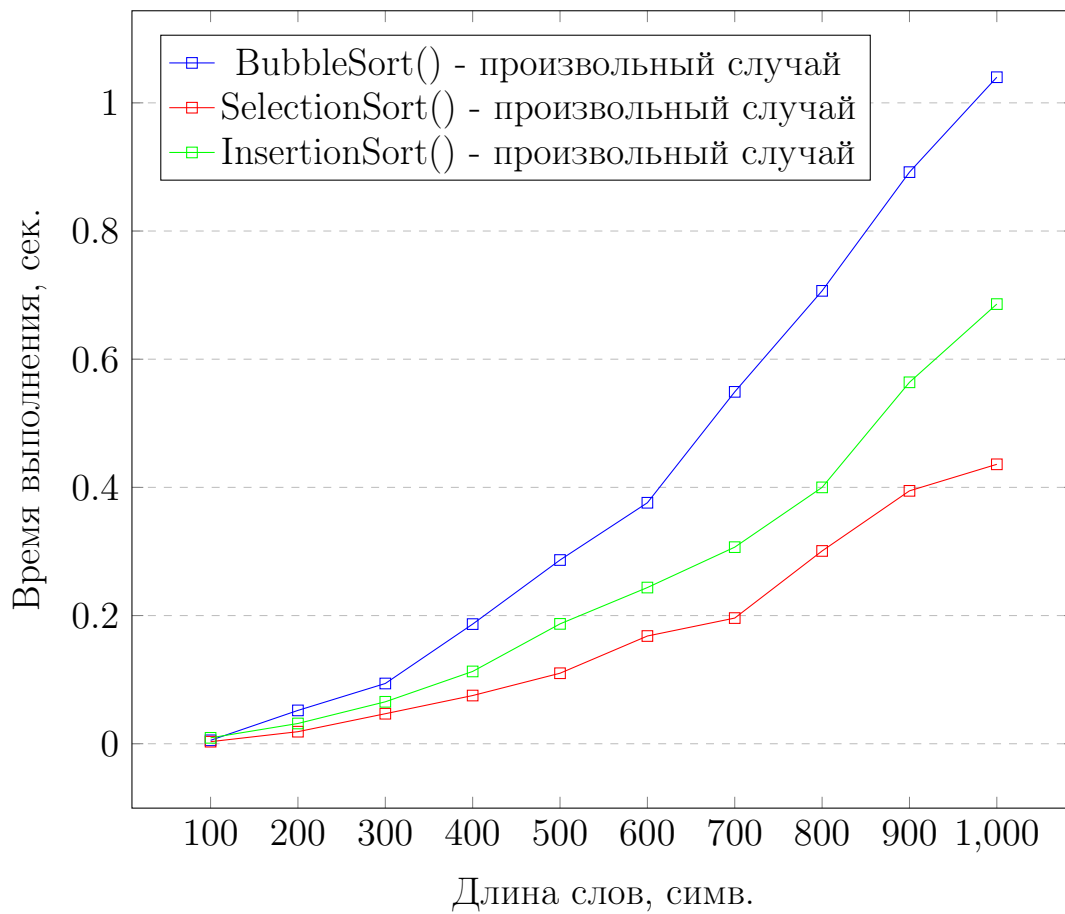


Рис. 4.7: Зависимость времени выполнения алгоритмов сортировок от длины массива в произвольном случае

Также приведены графики (рисунки 4.7-4.9) для сравнения алгоритмов сортировок между собой в произвольном, лучшем и худшем случаях.

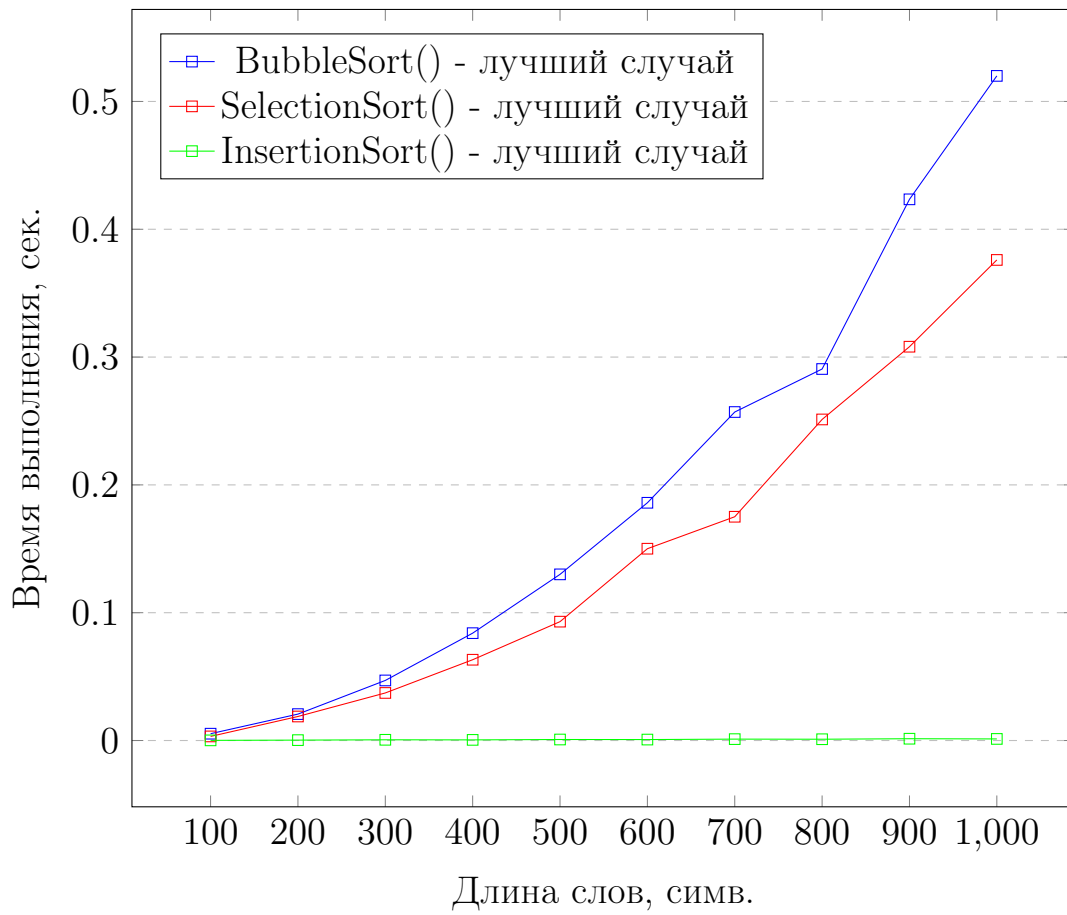


Рис. 4.8: Зависимость времени выполнения алгоритмов сортировок от длины массива в лучшем случае

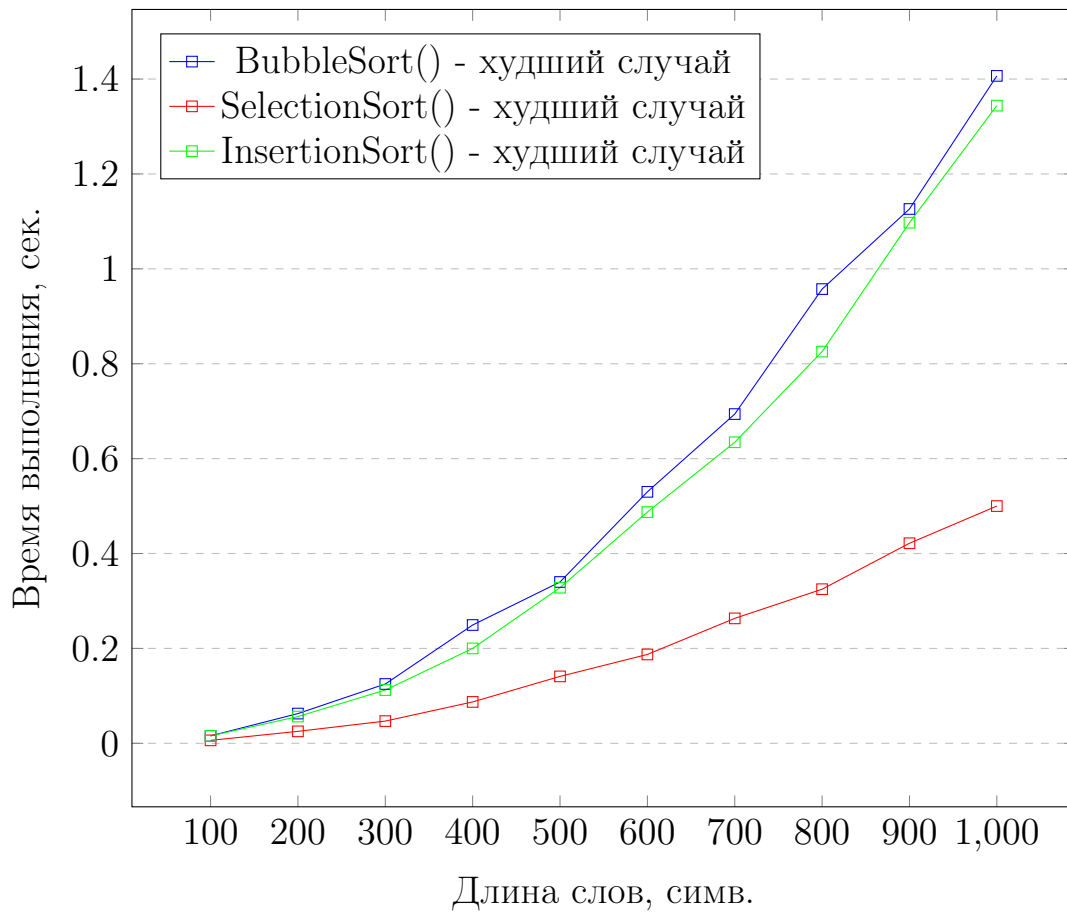


Рис. 4.9: Зависимость времени выполнения алгоритмов сортировок от длины массива в худшем случае

## 4.3 Оценка трудоёмкости

Для оценки трудоёмкости использовалась следующая модель вычислений: [4]

- Трудоёмкость следующих операций единична: +, -, =, +=, -=, ==, !=, <, >, <=, >=, «, », [];
- Трудоёмкость следующих операций = 2: \*, /, %, /=, \*=.

Трудоёмкость выбранных алгоритмов сортировок рассчитывалась по написанному коду.

На листинге 4.1 представлена программа для вычисления трудоёмкости алгоритма сортировки пузырьком для худшего случая.

Листинг 4.1: Вычисление трудоёмкости алгоритма сортировки пузырьком

```
1 int getBubbleSort(int *l, int *r)
2 {
3     int rez = 3; //init+srav
4     for (int i = 0; i < r-l; i++)
5     {
6         rez += 3; //init+srav
7         for (int *j = l; j < r-i; j++)
8         {
9             if (*j > *(j+1))
10                swap(j, (j+1));
11            rez+=5; //telo j
12            rez++; //increment
13            rez+=2; //srav
14        }
15        rez++; //increment
16        rez+=2; //srav
17    }
18    return rez;
19 }
```

Соответственно получается следующая формула трудоёмкости:

$$F_{bubblesort} = 3 + (3 * (N - 1) + ((N - 1) * N / 2) * 8) + 3 * (N - 1),$$

где N - размер массива

В лучшем же случае, не надо будет менять элементы местами, а значит трудоёмкость тела цикла по j уменьшится на 3 и формула примет вид:

$$F_{bubble\_sort} = 3 + (3 * (N - 1) + ((N - 1) * N / 2) * 5) + 3 * (N - 1)$$

На листинге 4.2 представлена программа для вычисления трудоёмкости алгоритма сортировки выбором для худшего случая.

Листинг 4.2: Вычисление трудоёмкости алгоритма сортировки выбором

```
1 int getSelectionSort(int *l, int *r)
2 {
3     int rez = 2; //init+srav
4     for (int *i = l; i <= r; i++)
5     {
6         rez += 2; //double = (assignment)
7         int minz = *i, *ind = i;
8         rez += 3; //init+srav
9         for (int *j = i + 1; j <= r; j++)
10        {
11            if (*j < minz)
12            {
13                minz = *j;
14                ind = j;
15            }
16            rez += 3; //telo j
17            rez++; //increment
18            rez++; //srav
19        }
20        rez+=3; //swap
21        swap(i, ind);
22        rez++; //increment
23        rez++; //srav
24    }
25    return rez;
26 }
```

Следовательно формула трудоёмкости будет следующей:

$$F_{selection\_sort} = 2 + 10 * N + ((N - 1) * N/2) * 5$$

А в лучшем случае не будет выполняться условие if и формула станет такой:

$$F_{selection\_sort} = 2 + 10 * N + ((N - 1) * N/2) * 3$$

На листинге 4.3 представлена программа для вычисления трудоёмкости алгоритма сортировки вставками для худшего случая.

Листинг 4.3: Вычисление трудоёмкости алгоритма сортировки вставками

```
1 int getInsertionSort(int* l, int* r)
2 {
3     int rez = 3; //init+srav
4     for (int *i = l + 1; i <= r; i++)
5     {
6         rez++; //assigment
7         int* j = i;
8         rez+=3; //srav
9         while (j > l && *(j - 1) > *j)
10        {
11            rez+=4; //swap
12            swap((j - 1), j);
13            j--;
14            rez++; //deccrement
15            rez+=3; //srav
16        }
17    }
18    return rez;
19 }
```

Формула трудоёмкости:

$$F_{insertion\_sort} = 3 + 4 * (N - 1) + 8 * ((N - 1) * N / 2)$$

В лучшем случае полностью пропадает тело цикла while, а значит формула изменится на следующую:

$$F_{insertion\_sort} = 3 + 4 * (N - 1)$$

## 4.4 Вывод

По итогу исследования выяснилось, что разработанная программа работает верно, то-есть сортирует массивы по возрастанию. Кроме этого, смотря на время выполнения каждого алгоритма, логично сделать вывод, что наиболее быстрым в произвольном случае, является алгоритм сортировки выбором и судя по оценке трудоёмкости, наименее трудоёмким является также алгоритм сортировки выбором.



# Заключение

По итогу проделанной работы была достигнута цель - изучены алгоритмы сортировки и получены навыки оценки трудоемкости алгоритмов.

Также были решены все поставленные задачи, а именно:

- реализованы 3 выбранных алгоритма сортировки;
- выполнена оценка времени выполнения алгоритмов сортировки;
- рассчитана трудоемкость каждого из алгоритма сортировки.

## Список использованных источников

- [1] Опанасенко М. Описание алгоритмов сортировки и сравнение их производительности [Электронный ресурс]. // URL: <https://habr.com/ru/post/335920/>.
- [2] Getprocesstimes function (processthreadsapi.h) [Электронный ресурс]. // URL: <https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesstimes#syntax>.
- [3] Как применить настройки оптимизации gcc в qt? // URL: <http://blog.kislenko.net/show.php?id=1991>.
- [4] Ульянов М.В. *Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ*. Москва: ФИЗМАТЛИТ, 2008. 304 с.