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

Введение

Цель работы – изучение алгоритмов сортировки и оценка их трудоемкости.

Задачи работы:

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;
- соритровка 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 находиться 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. Нужно отметить, что эту сортировку можно реализовать двумя способами – сохраняя минимум и его индекс или просто переставляя текущий элемент с рассматриваемым, если они стоят в неправильном порядке. Далее будет реализован и рассмотрен первый способ. [1]

1.3 Сортировка вставками

Создаётся массив, в котором после завершения алгоритма будет лежать 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): просто сделаем так, чтобы отсортирован был некоторый префикс исходного массива, вместо вставки будем менять текущий элемент с предыдущим, пока они стоят в неправильном порядке. [1]

2 Конструкторский раздел

В разделе представлены схемы следующих алгоритмов сортировки:

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

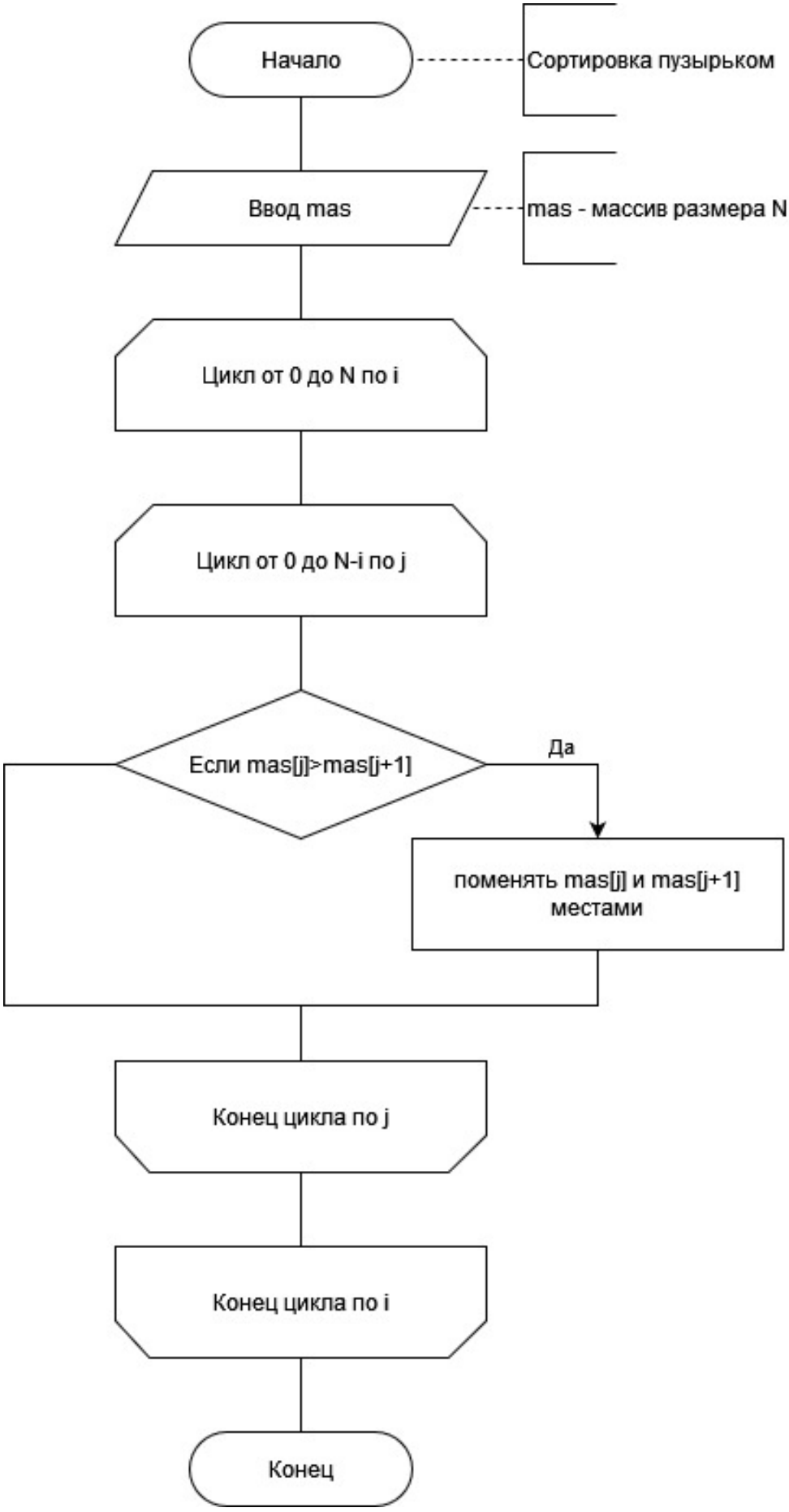

Рис. 2.1: Сортировка пузырьком

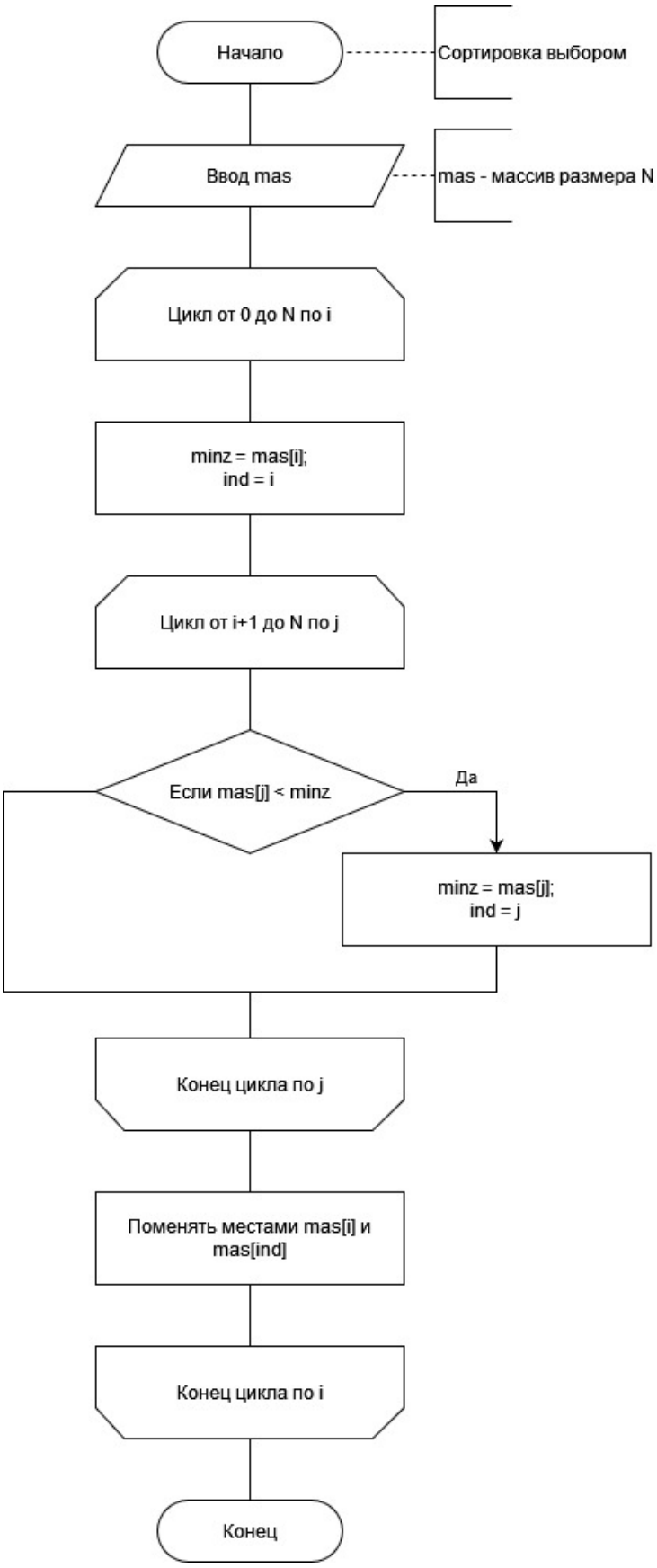

Рис. 2.2: Сортировка выбором

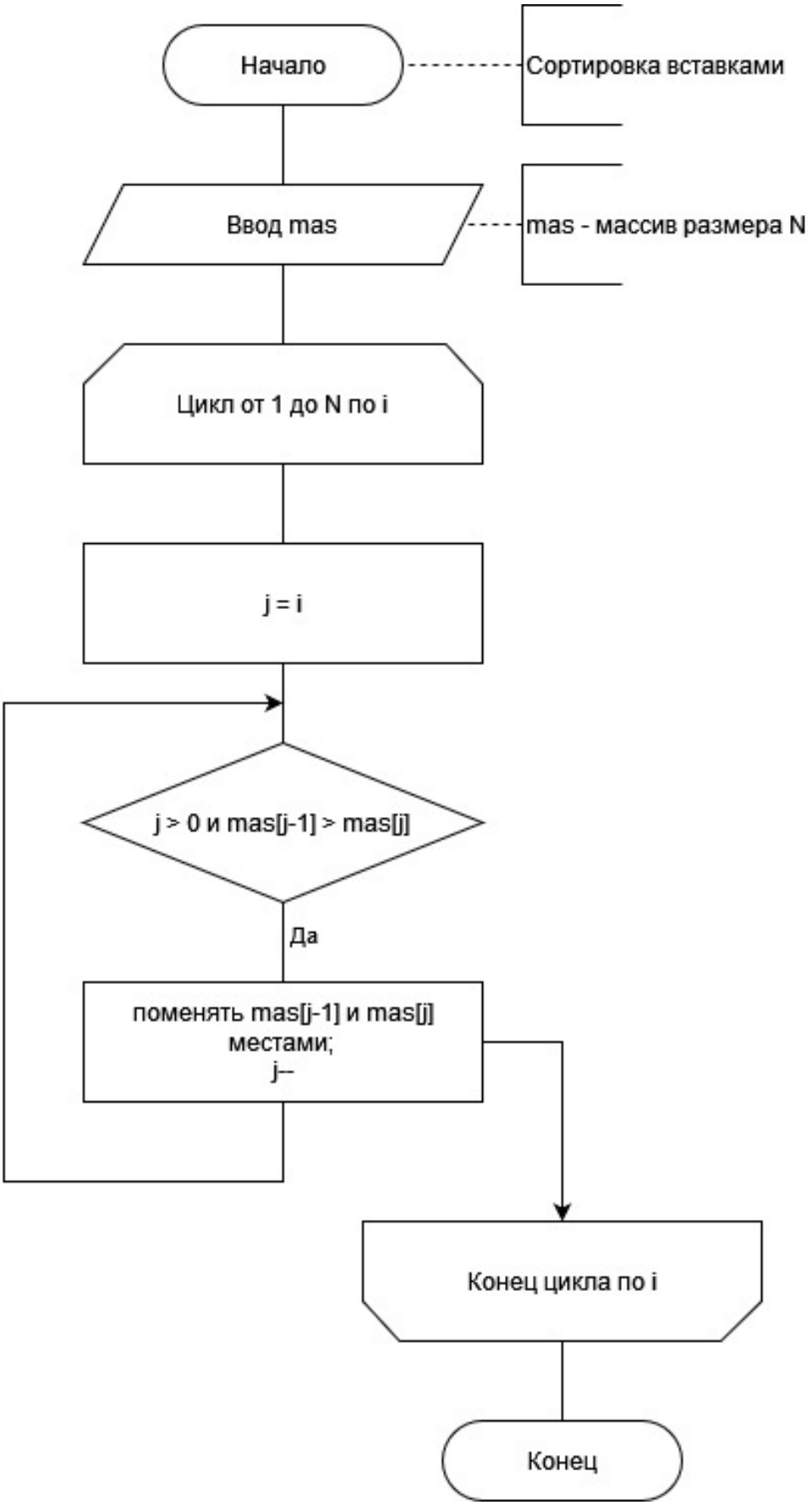

Рис. 2.3: Сортировка вставками

3 Технологический раздел

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

3.1 Выбор инструментов

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

Чтобы оценить время выполнения программы будет замерятся процессорное время, т.к. такиим образом будут получены данные подходящие для целесообразного сравнения алгоритмов. Для замера процессорного времени программы используется функция GetProcessTimes() т.к. я пишу код под Windows. [2]

Кроме этого, необходимо отключить оптимизации компилятора для более честного сравнения алгоритмов. В моём случае это делается с помощтю ключа $-O0$. [3]

Для оценки памяти, используемой программой был выбран valgrind. Для его функционирования была установлена wsl (Windows Subsystem for Linux) и виртуальная машина с Ubuntu. Для замера максимального кол-ва памяти используемой алгоритмами созданы отдельные файлы, которые выполняют отдельно взятый алгоритм с массивами определённой длины. Затем они компилируется с помощью g++ и получается информация об используемой памяти с помощью valgrind –tool=massif. [4]

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

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

Листинг 3.2: Реализация алгоритма сортировки выбором

```
void SelectionSort(int *l, int *r)
{
        for (int *i = l; i <= r; i++)
        {
                int minz = *i, *ind = i;
                for (int *j = i + 1; j <= r; j++)
                {
                        if (*j < minz)
                        {
                                minz = *j;
                                ind = j;
                        }
                }
                swap(i, ind);
        }
}
```

Листинг 3.3: Реализация алгоритма сортировки вставками

```
void InsertionSort(int* l, int* r)
{
        for (int *i = l + 1; i <= r; i++)
        {
                int* j = i;
                while (j > l && *(j - 1) > *j)
                {
                        swap((j - 1), j);
                        j--;
                }
        }
}
```

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.

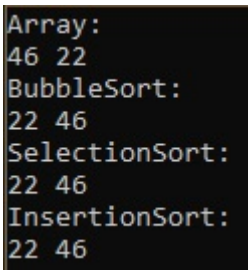

Рис. 4.1: Результаты сортировки массива с размером = 2

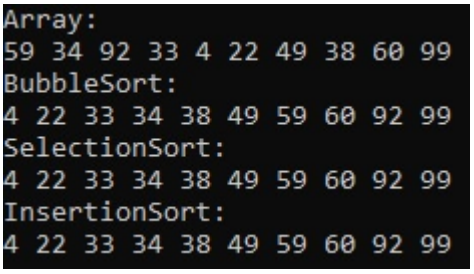

Рис. 4.2: Результаты сортировки массива с размером = 10

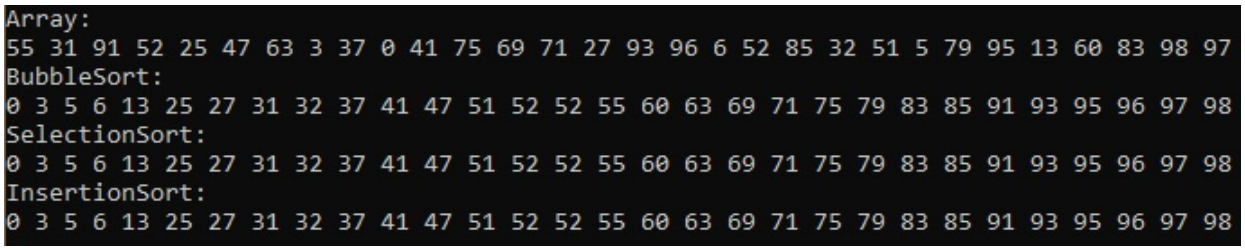

Рис. 4.3: Результаты сортировки массива с размером = 30

4.2 Сравнительный анализ времени выполнения алгоритмов

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.

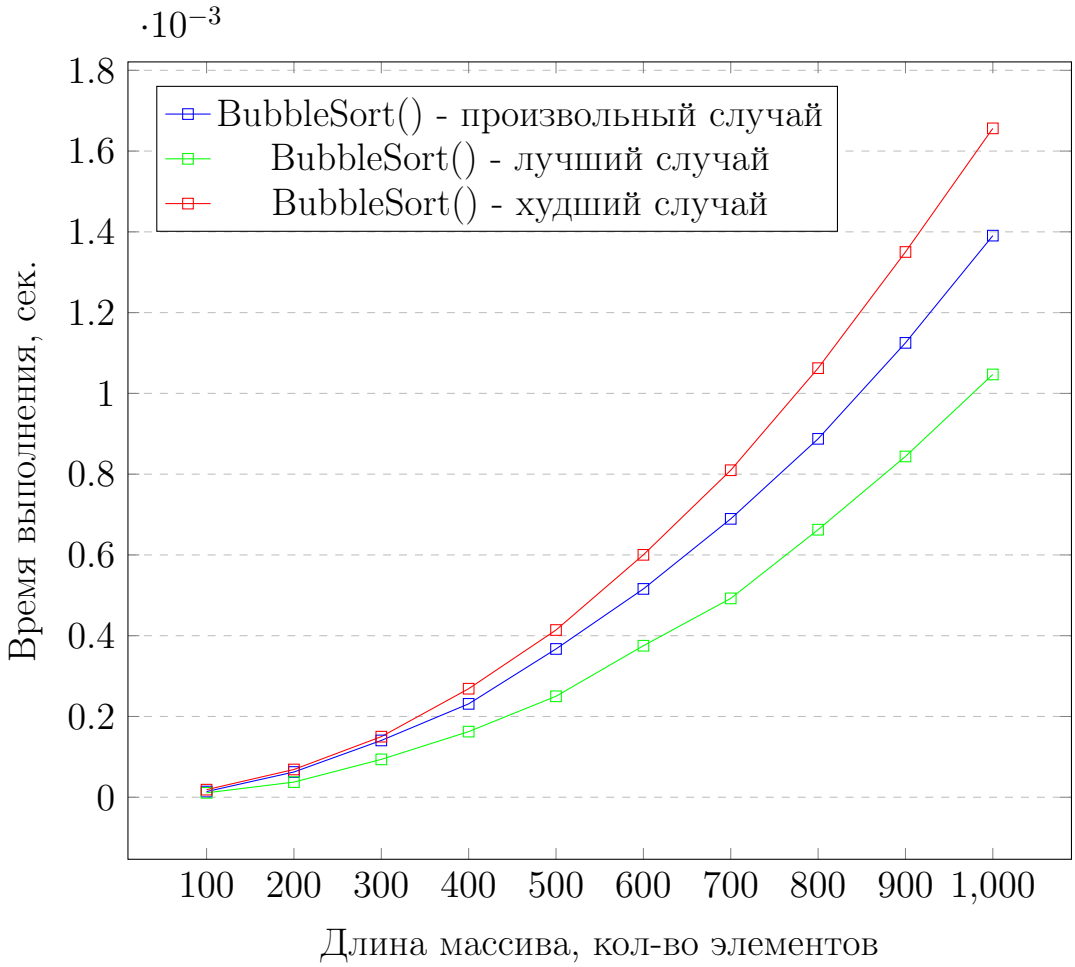

Рис. 4.4: Зависимость времени выполнения сортировки пузырьком от длины массива в разных случаях

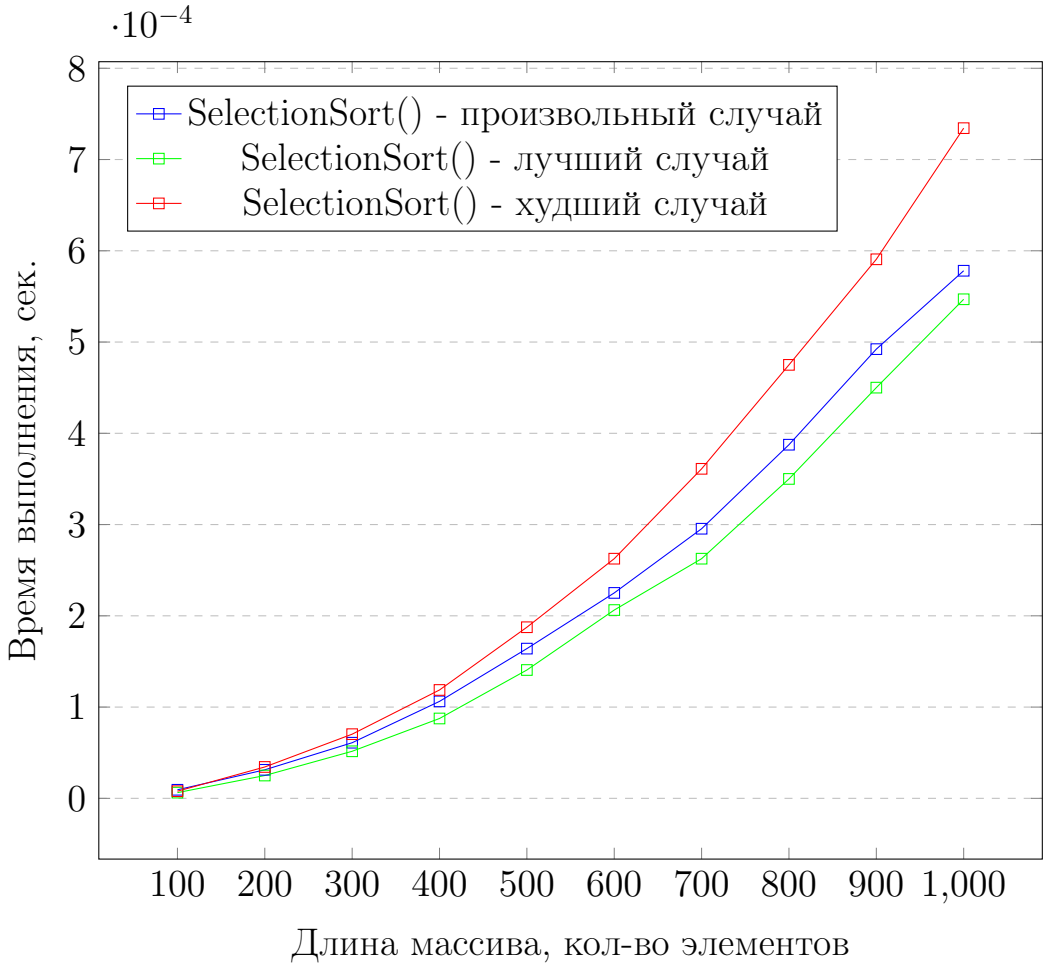

Рис. 4.5: Зависимость времени выполнения сортировки выбором от длины массива в разных случаях

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.

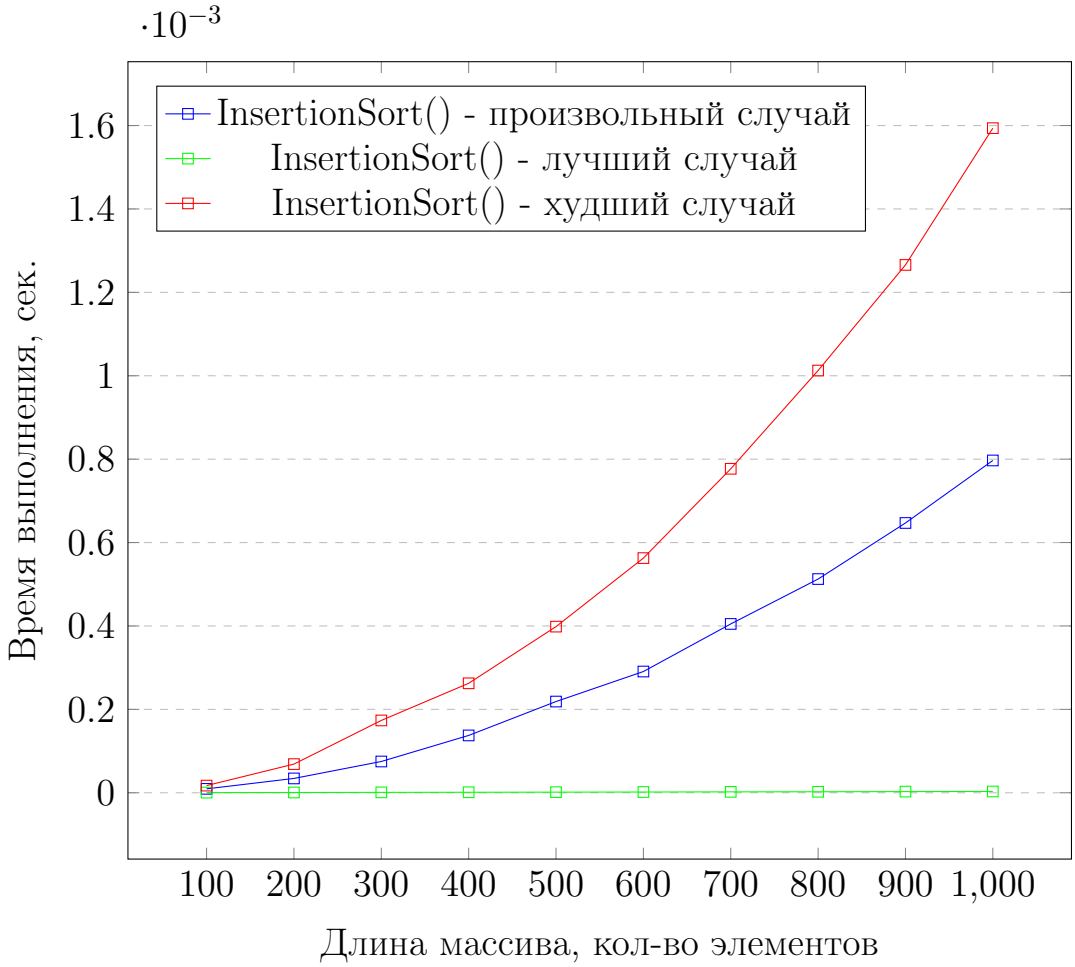

Рис. 4.6: Зависимость времени выполнения сортировки выбором от длины массива в разных случаях

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.

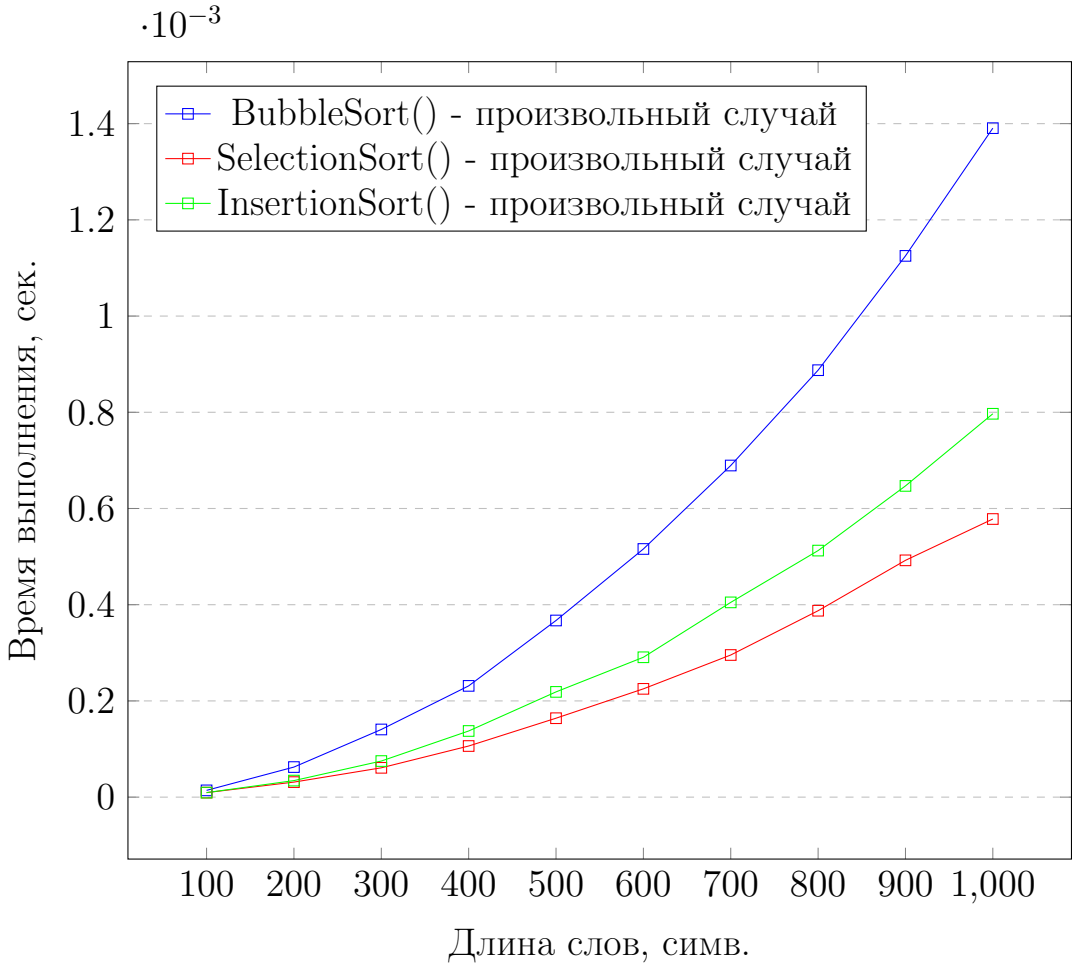

Рис. 4.7: Зависимость времени выполнения алгоритмов сортировок от длины массива в произвольном случае

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

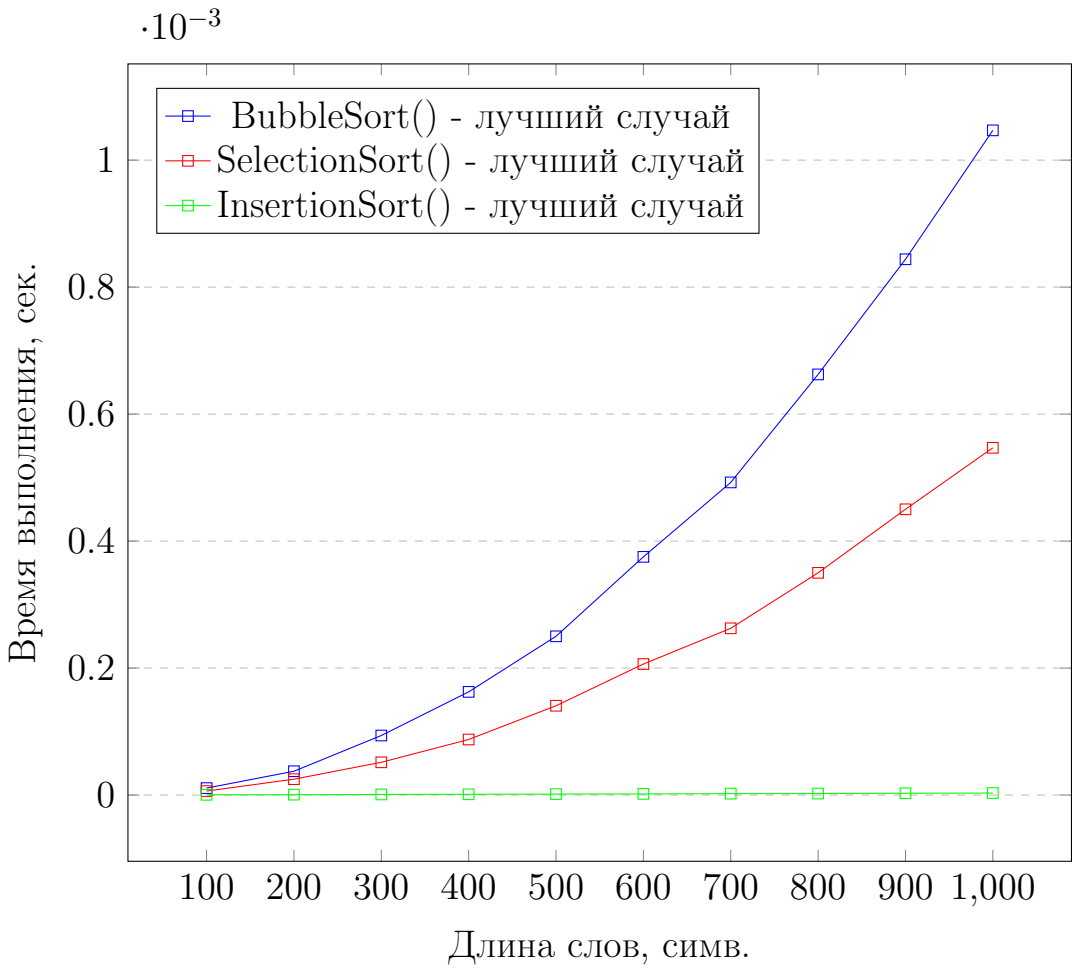

Рис. 4.8: Зависимость времени выполнения алгоритмов сортировок от длины массива в лучшем случае

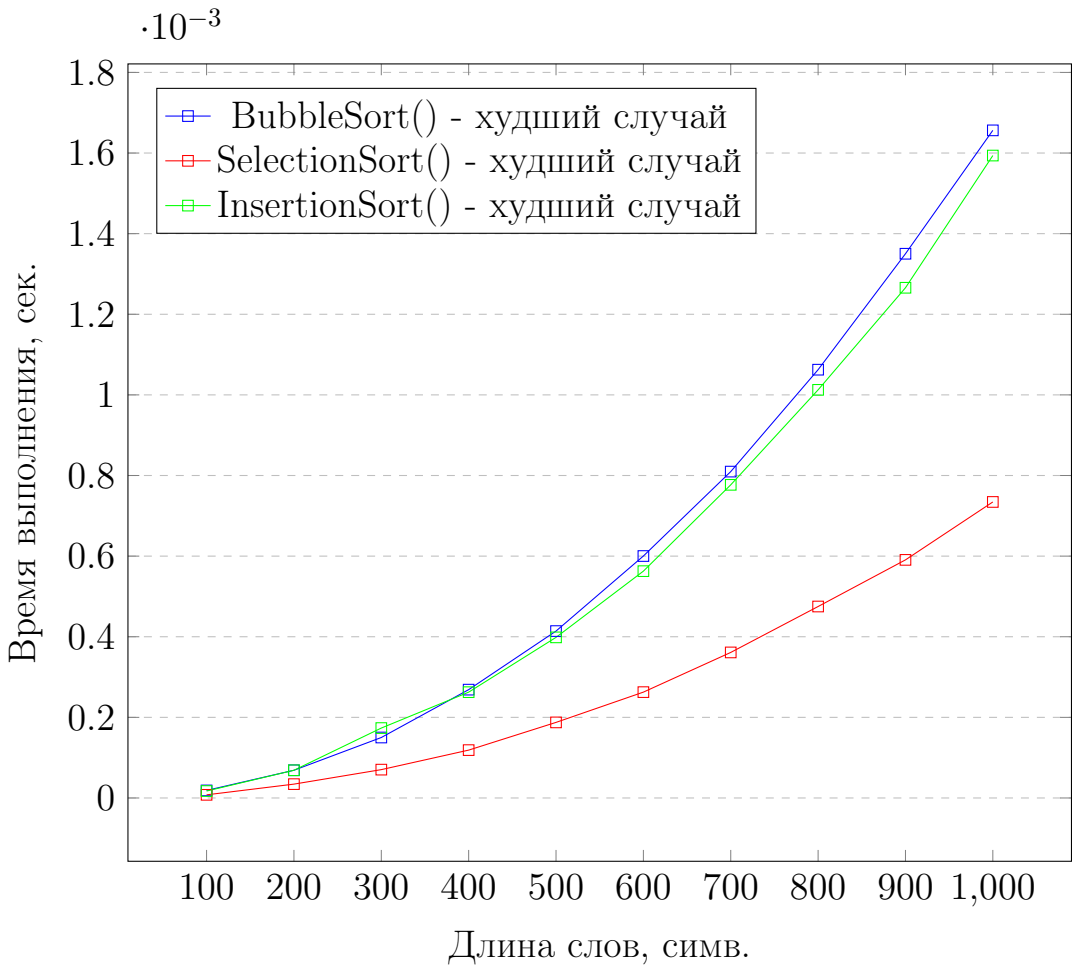

Рис. 4.9: Зависимость времени выполнения алгоритмов сортировок от длины массива в худшем случае

4.3 Оценка трудоёмкости

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

Листинг 4.1: Вычисление трудоёмкости алгоритма сортировки пузырьком

```
int getBubbleSort(int *l, int *r)
{
        int rez = 3; //init+srav
        for (int i = 0; i < r-l; i++)
        {
                rez += 3; //init+srav
                for (int *j = l; j < r-i; j++)
                {
                        if (*j > *(j+1))
                                swap(j, (j+1));
                        rez+=5; //telo j
                        rez++; //increment
                        rez+=2; //srav
                }
                rez++; //increment
                rez+=2; //srav
        }
        return rez;
}
```

Соответсnвенно получается:

$Fbubblesort = 3 + (3 * (N-1) + ((N-1) * N/2) * 8) + 3 * (N-1)$

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

$Fbubble\_sort = 3 + (3 * (N-1) + ((N-1) * N/2) * 5) + 3 * (N-1)$

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

Листинг 4.2: Вычисление трудоёмкости алгоритма сортировки выбором

```
int getSelectionSort(int *l, int *r)
{
        int rez = 2; //init+srav
        for (int *i = l; i <= r; i++)
        {
                rez += 2; //double = (assignment)
                int minz = *i, *ind = i;
                rez += 3; //init+srav
                for (int *j = i + 1; j <= r; j++)
                {
                        if (*j < minz)
                        {
                                minz = *j;
                                ind = j;
                        }
                        rez += 3; //telo j
                        rez++; //increment
                        rez++; //srav
                }
                rez+=3; //swap
                swap(i, ind);
                rez++; //increment
                rez++; //srav
        }
        return rez;
}
```

Следовательно формула трудоёмкости будет следующей:

$$Fselection\_sort = 2 + 10 * N + ((N - 1) * N/2) * 5$$

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

$$Fselection\_sort = 2 + 10 * N + ((N - 1) * N/2) * 3$$

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

Листинг 4.3: Вычисление трудоёмкости алгоритма сортировки вставками

```
int getInsertionSort(int* l, int* r)
{
        int rez = 3; //init+srav
        for (int *i = l + 1; i <= r; i++)
        {
                rez++; //assigment
                int* j = i;
                rez+=3; //srav
                while (j > l && *(j - 1) > *j)
                {
                        rez+=4; //swap
                        swap((j - 1), j);
                        j--;
                        rez++; //deccrement
                        rez+=3; //srav
                }
        }
        return rez;
}
```

Формула трудоёмкости:

$Finsertion\_sort = 3 + 4 * (N - 1) + 8 * ((N - 1) * N/2)$

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

$Finsertion\_sort = 3 + 4 * (N - 1)$

4.4 Вывод

По итогу иследования выяснилось, что разработанная программа работает верно. Кроме этого, смотря на время выполнения каждого алгоритма, логично сделать вывод, что наиболее быстрым в произвольном случае, является алгоритм соритровки выбором.

Заключение

По итогу проделанной работы была достигнута цель - изучены алгоритмы сортировки и проведена оценка их трудоемкости.

Также были решены все поставленные задачи, а именно:

- реализованы...

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] Massif: a heap profiler [Электронный ресурс]. // URL: https://valgrind.org/docs/manual/ms-manual.html.