



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 Криков А.В.

Группа ИУ7-53Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Содержание

Введение	2
1 Аналитическая часть	4
1.1 Быстрая сортировка	4
1.2 Сортировка вставками	4
1.3 Сортировка выбором	4
2 Конструкторская часть	6
2.1 Разработка алгоритмов	6
2.2 Модель вычислений	8
2.3 Трудоёмкость алгоритмов	9
2.3.1 Алгоритм сортировки выбором	9
2.3.2 Алгоритм быстрой сортировки	10
2.3.3 Алгоритм сортировки вставками	10
2.3.4 Память используемая алгоритмом	10
2.3.5 Описание структур данных	11
2.3.6 Описание способов тестирования	11
3 Технологическая часть	12
3.1 Требования к ПО	12
3.2 Средства реализации	12
3.3 Листинг кода	12
3.4 Тестирование функций	14
4 Исследовательская часть	16
4.1 Технические характеристики	16
4.2 Время выполнения алгоритмов	16
Заключение	20

Введение

В данной лабораторной работе будут рассмотрены алгоритмы сортировки.

Алгоритмы сортировки используются практически в любой программной системе. Целью алгоритмов сортировки является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количеству элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное количеству элементов в последовательности, то есть намного больше. Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядоченность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Целью данной работы является изучение трех алгоритмов сортировки и реализации данных алгоритмов.

В рамках выполнения работы необходимо решить следующие задачи:

- исследовать и сравнить 3 алгоритма сортировки: вставками, выбором и алгоритм быстрой сортировки;
- привести схемы рассматриваемых алгоритмов (вставками, выбором, алгоритм быстрой сортировки);
- описать используемые структуры данных;
- оценить объем памяти для хранения данных;
- описать структуру разрабатываемого ПО.
- определить средства программной реализации;

- протестировать разработанное ПО.
- провести сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы сортировки, используемые для выполнения лабораторной работы.

1.1 Быстрая сортировка

В алгоритме быстрой сортировки (Quicksort) используется рекурсивный подход. Выбрав опорный элемент в списке данный алгоритм сортировки делит список на две части, относительно выбранного элемента. Далее в первую часть попадают все элементы, меньшие выбранного, а во вторую — большие элементы. Если в данных частях более двух элементов, рекурсивно запускается для него та же процедура. В конце получится полностью отсортированная последовательность.

1.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, которым элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [1].

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

1.3 Сортировка выбором

Шаги алгоритма:

1. находим номер минимального значения в текущем массиве;

2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
3. теперь сортируем “хвост” массива, исключив из рассмотрения уже отсортированные элементы.

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов, что может привести к резкому увеличению числа обменов.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: вставками, выбором и быстрой сортировки. Входными данными будут является массив, а также его размерность. Выходными данными будет являться отсортированный массив. В связи с ограничениями на ПО, массив должен содержать только целые числа. Все элементы массива, а также его размерность должны быть валидными. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов сортировки.

2.1 Разработка алгоритмов

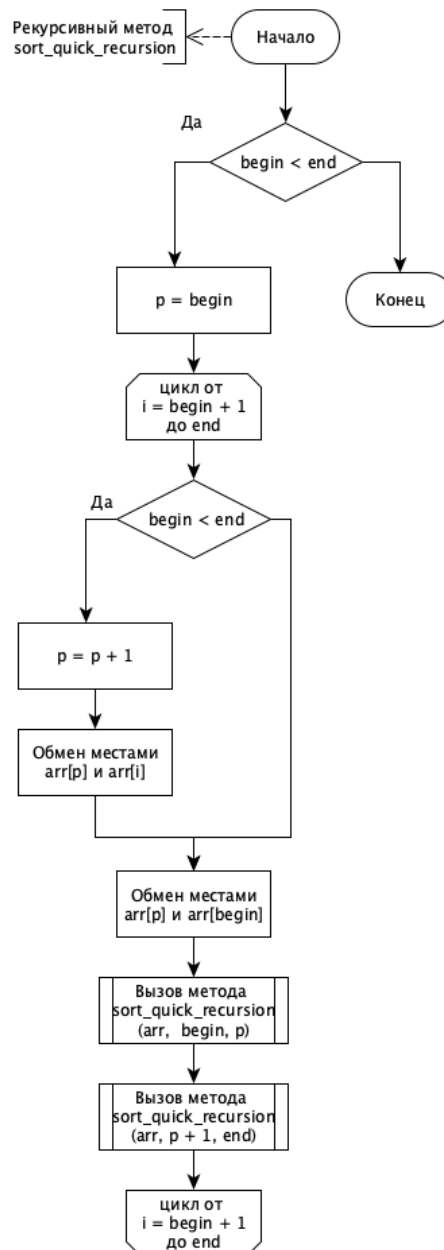


Рис. 2.1: Схема алгоритма быстрой сортировки

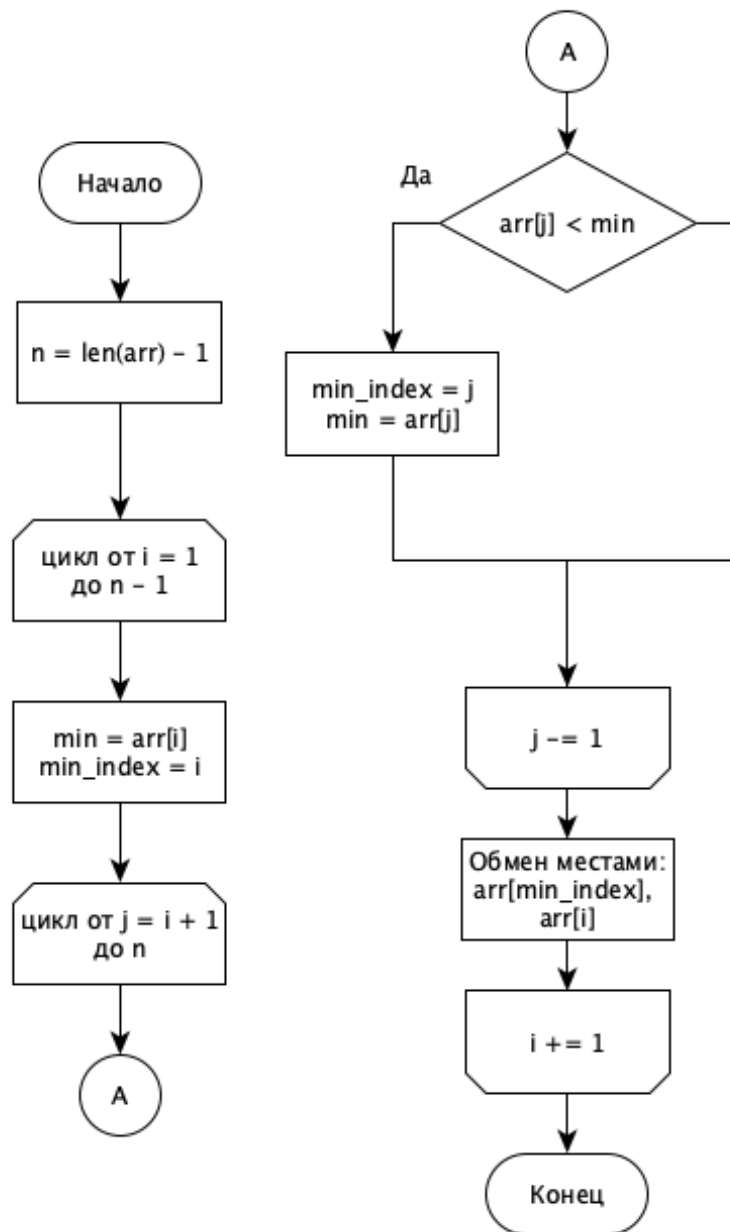


Рис. 2.2: Схема алгоритма сортировки выбором

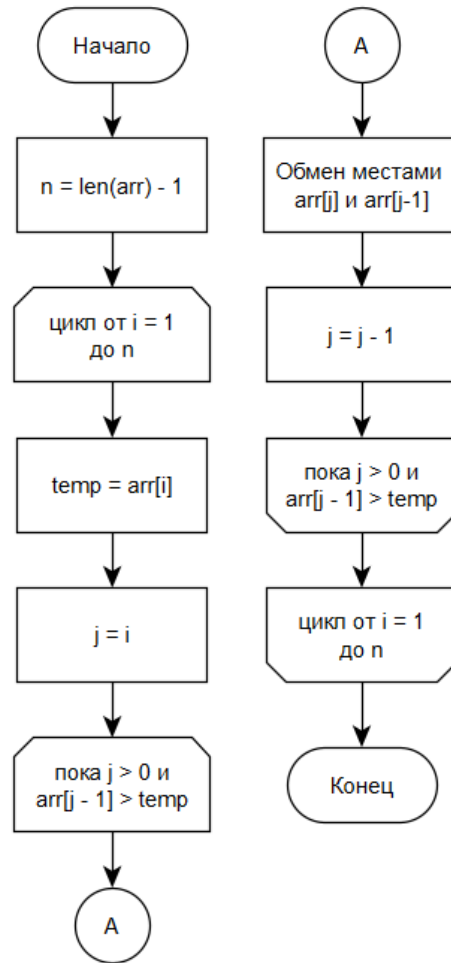


Рис. 2.3: Схема алгоритма сортировки вставками

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, *, \%, =, + =, - =, * =, / =, \% =, ==, !=, <, >, <=, >=, [], ++, -- \quad (2.1)$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{условия} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инкремента} + f_{сравнения}) \quad (2.3)$$

4. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Обозначим во всех последующих вычислениях размер массивов как N .

2.3.1 Алгоритм сортировки выбором

Трудоёмкость алгоритма сортировки выбором состоит из:

- Трудоёмкость сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [1..N]$, которая равна (2.4):

$$f_{outer} = 2 + 12(N - 1) \quad (2.4)$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$, которая равна (2.5):

$$f_{inner} = 2(N - 1) + \frac{N \cdot (N - 1)}{2} \cdot f_{if} \quad (2.5)$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 3, & \text{х.с.} \end{cases} \quad (2.6)$$

Трудоёмкость в лучшем случае (2.7):

$$f_{best} = -12 + 12.5N + \frac{3}{2}N^2 \approx \frac{3}{2}N^2 = O(N^2) \quad (2.7)$$

Трудоёмкость в худшем случае (2.8):

$$f_{worst} = -12 + 11N + 3N^2 \approx 3N^2 = O(N^2) \quad (2.8)$$

2.3.2 Алгоритм быстрой сортировки

Трудоёмкость в лучшем случае (2.9):

$$f_{best} = O(n * \log(N)) \quad (2.9)$$

Трудоёмкость в худшем случае (2.10):

$$f_{worst} = O(N^2) \quad (2.10)$$

2.3.3 Алгоритм сортировки вставками

Трудоёмкость в лучшем случае (2.11):

$$f_{best} = O(n) \quad (2.11)$$

Трудоёмкость в худшем случае (2.12):

$$f_{worst} = O(N^2) \quad (2.12)$$

2.3.4 Память используемая алгоритмом

Сортировки выбором и вставками не используют дополнительных структур, поэтому затраченная память будет равна $O(n)$. Быстрая сортировка использует $O(\log(n))$ дополнительной памяти, хранящейся в стеке. Полная затраченная память будет равна $O(n) + O(\log(n))$.

2.3.5 Описание структур данных

Для реализации алгоритмов сортировки использовалась структура данных - массив. Также можно было сортировать связный список, однако при сортировке данной структуры потребуется гораздо больше времени, поэтому был выбран массив.

2.3.6 Описание способов тестирования

Данные сортировки можно протестировать функционально. При этом можно выделить следующие классы эквивалентности:

- Упорядоченный массив;
- Массив одинаковых элементов;
- Неупорядоченный массив;
- Массив из одного элемента;

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов;
- на выходе — тот же массив, но в отсортированном порядке.

3.2 Средства реализации

В данной лабораторной работе использовался язык программирования - С. [2] Данный язык быстр и удобен в использовании. В качестве среды разработки использовался Visual Studio Code. [3]

3.3 Листинг кода

В листингах 3.1 – 3.3 приведены листинги алгоритма сортировки вставками, выбором и алгоритм быстрой сортировки соответственно. В листинге 3.4 приведен пример реализации бенчмарков.

Листинг 3.1: Алгоритм сортировки вставками

```
1 void insertion_sort(int n, int array[])
2 {
3     int new_elem, location;
4
5     for (int i = 1; i < n; i++)
6     {
7         new_elem = array[i];
8         location = i - 1;
9         while(location >= 0 && array[location] > new_elem)
10        {
11            array[location+1] = array[location];
12            location = location - 1;
```

```

13     }
14     array[location+1] = new_elem;
15 }
16 }

```

Листинг 3.2: Алгоритм сортировки выбором

```

1 void choice_sort(int *arr, int n)
2 {
3     int min_pos, tmp;
4     for (int i = 0; i < n; i++)
5     {
6         min_pos = i;
7         for (int j = i + 1; j < n; j++)
8             if (arr[min_pos] > arr[j])
9                 min_pos = j;
10
11         tmp = arr[min_pos];
12         arr[min_pos] = arr[i];
13         arr[i] = tmp;
14     }
15 }

```

Листинг 3.3: Алгоритм быстрой сортировки

```

1 void quick_sort(int *array, int left, int right)
2 {
3     int pivot = array[left];
4     int l_hold = left;
5     int r_hold = right;
6
7
8     while (left < right)
9     {
10         while ((array[right] >= pivot) && (left < right))
11             right--;
12
13         if (left != right)
14         {
15             array[left] = array[right];
16             left++;
17         }
18
19         while ((array[left] <= pivot) && (left < right))
20             left++;
21
22         if (left != right)
23         {
24             array[right] = array[left];

```

```

25         right--;
26     }
27 }
28
29 array[left] = pivot;
30 pivot = left;
31 left = l_hold;
32 right = r_hold;
33
34 if (left < pivot)
35     quick_sort(array, left, pivot - 1);
36 if (right > pivot)
37     quick_sort(array, pivot + 1, right);
38 }

```

Листинг 3.4: Пример реализации бенчмарка

```

1 double count_time_insertion_sort(int array[], int n)
2 {
3     clock_t start, end;
4
5     start = clock();
6     insertion_sort(n, array);
7     end = clock();
8
9     return (double)(end - start) / CLOCKS_PER_SEC;
10 }

```

3.4 Тестирование функций

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Входной массив	Ожидаемый результат	Результат
[1,2,3,4]	[1,2,3,4]	[1,2,3,4]
[3,2,1]	[1,2,3]	[1,2,3]
[5,6,2,4, - 2]	[-2,2,4,5,6]	[-2,2,4,5,6]
[4]	[4]	[4]
[]	[]	[]

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

В данном разделе будет произведено сравнение вышеизложенных алгоритмов.

4.1 Технические характеристики

- Операционная система: Windows 10 64 bit. [4]
- Память: 16 GiB.
- Процессор: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz [5].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Для сравнения возьмем массивы размерностью [100, 200, 300, ... 1000]. Так как подсчет сортировки массива считается короткой задачей, воспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма n раз, после чего поделим на n . Тем самым достаточно точные характеристики времени. Сравнение произведем при $n = 100$.

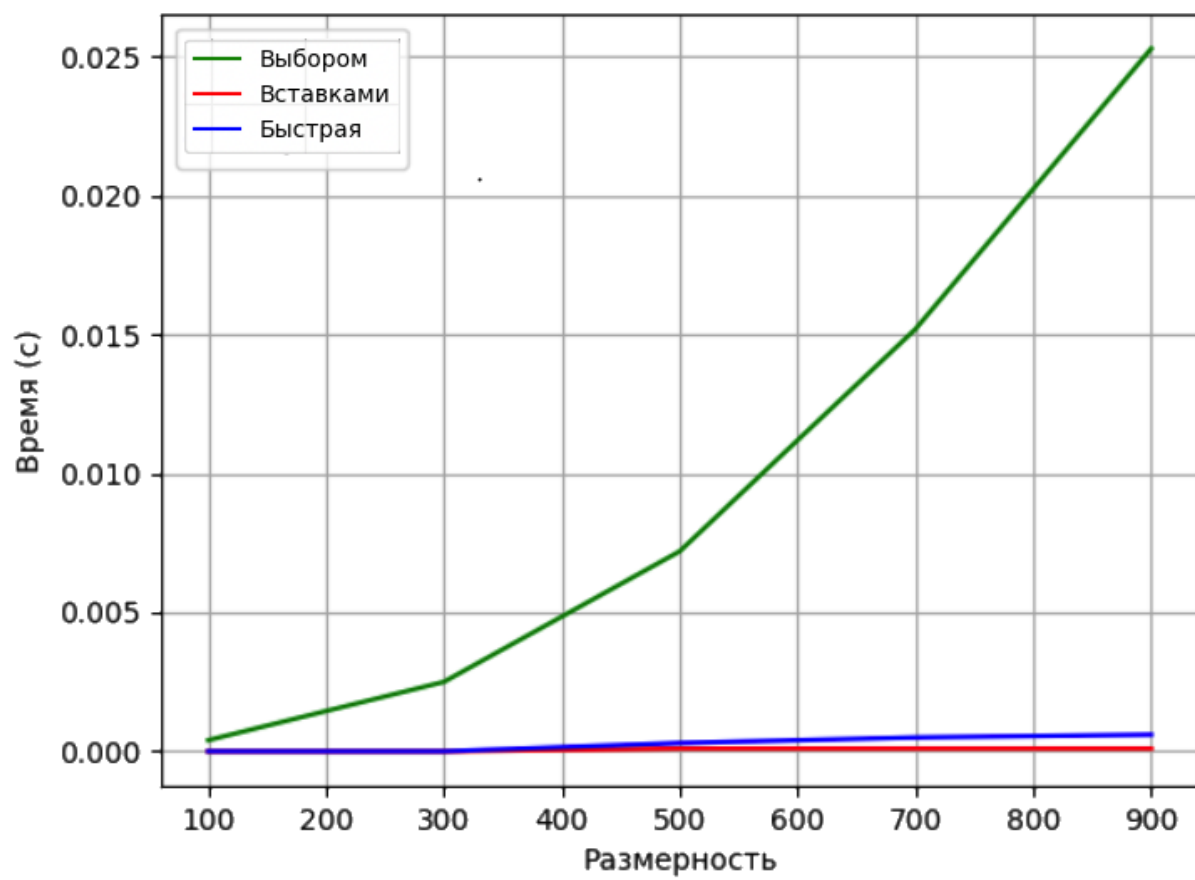


Рис. 4.1: Время работы алгоритмов на лучших данных

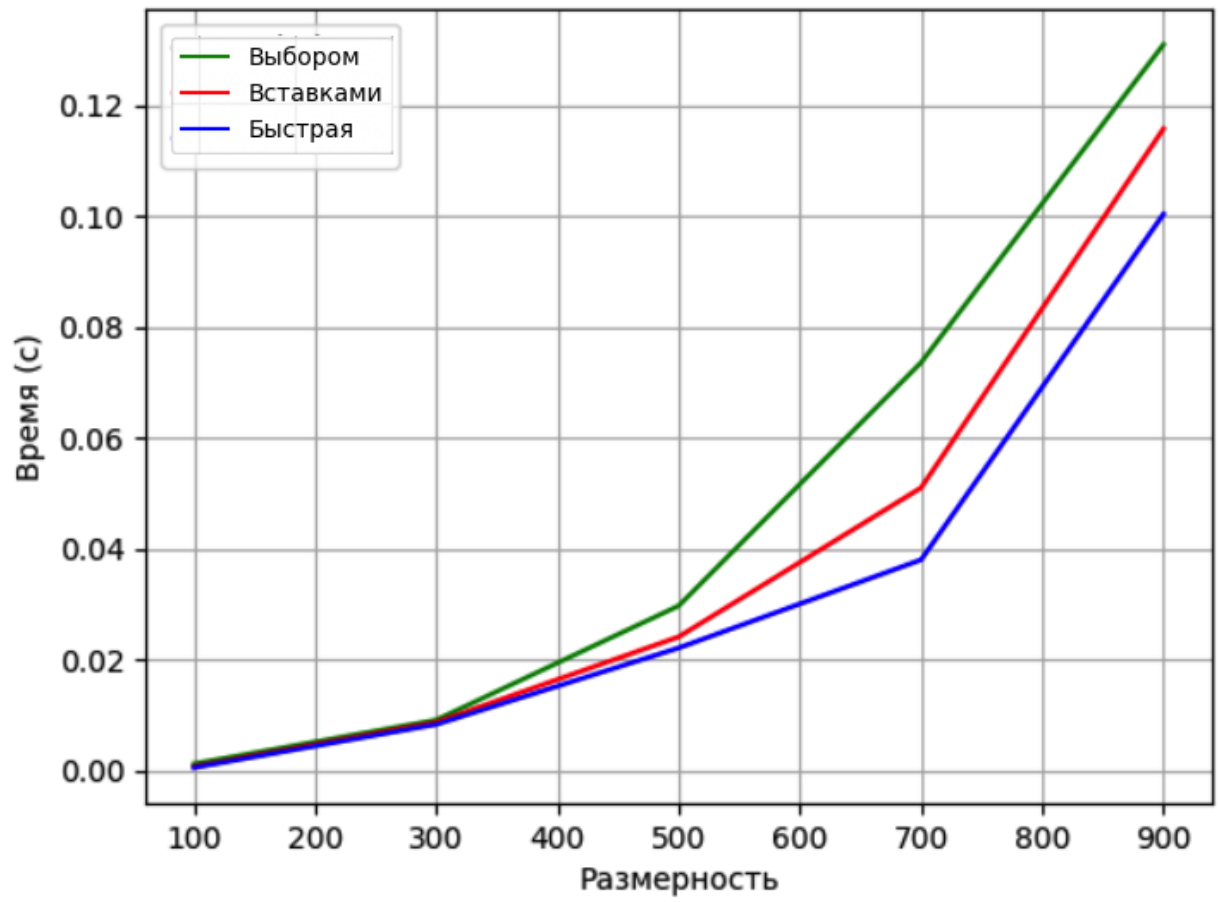


Рис. 4.2: Время работы алгоритмов на худших данных

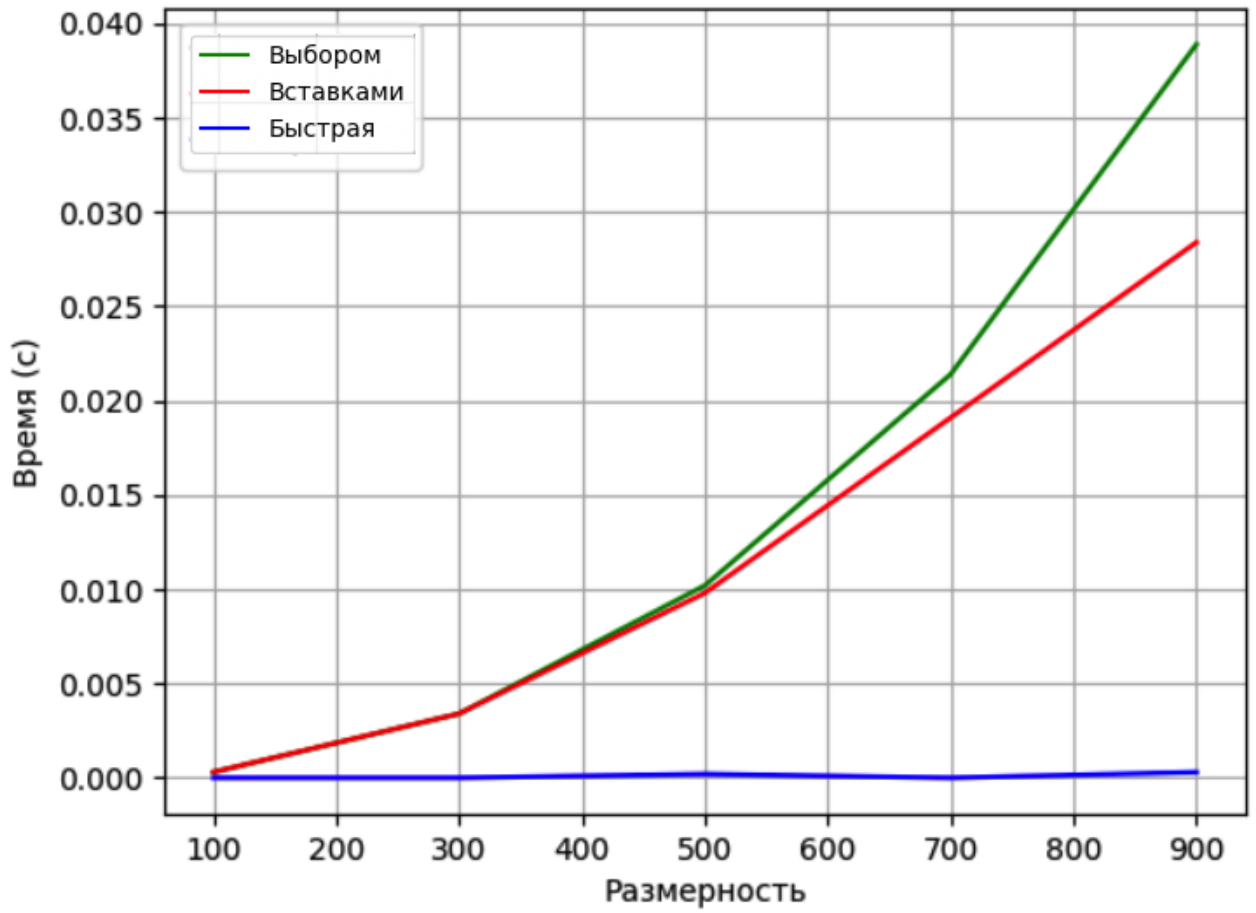


Рис. 4.3: Время работы алгоритмов на случайных данных

Вывод

В результате эксперимента было получено, что на случайных данных, алгоритм быстрой сортировки работает быстрее чем алгоритм выбором и вставками. Например при размерности массива равной 900 алгоритм быстрой сортировки работает в 8 раз быстрее алгоритма сортировки выбором и в 5.5 раз быстрее алгоритма вставками.

Заключение

В рамках лабораторной работы:

- были изучены и реализованы 3 алгоритма сортировки: вставками, выбором и быстрая сортировка;
- был проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- был подготовлен отчет по проделанной работе.

Анализируя результат проведенных экспериментов, приходим к выводу, что наиболее эффективной является быстрая сортировка. Также важно отметить, что на лучших данных эффективнее всего оказалась сортировка вставками.

Список литературы

- [1] Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования*. с. 834.
- [2] Брайан Керниган Деннис Ритчи. Язык программирования Си. Деннис Ритчи, 1978. Т. 1 из *Введение*. с. 9.
- [3] Documentation for Visual Studio Code [Электронный ресурс]. Режим доступа: <https://code.visualstudio.com/docs>.
- [4] Клиентская документация по Windows для ИТ-специалистов [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/windows/resources/>.
- [5] Процессор Intel® Core™ i7-4700HQ [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html>.