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.

Москва — 2021 г.

Оглавление

Введение

В данной лабораторной работе будут рассмотрены алгоримты 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:

- изучить и реализовать 3 алгоритма сортировки;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

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, котором 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. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

2.1 Разработка алгоритмов

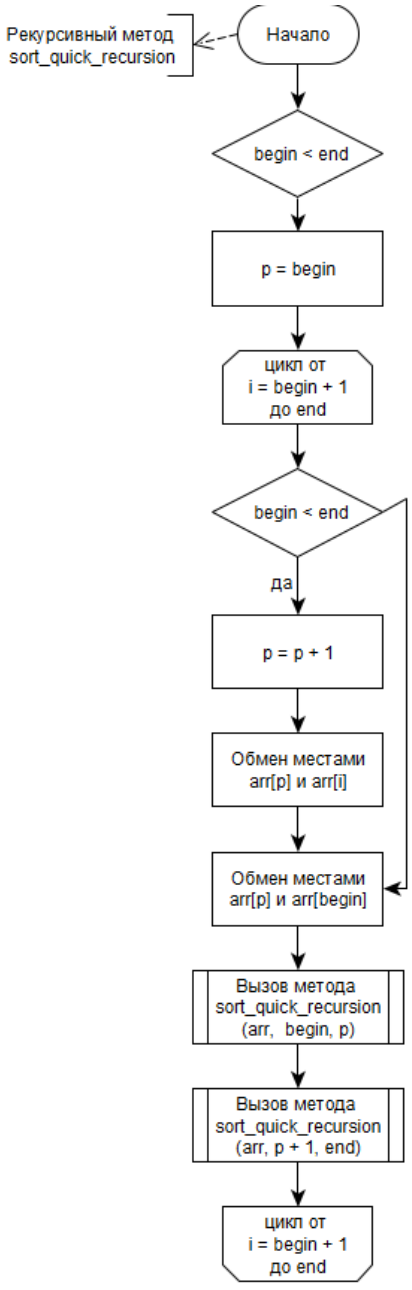

Рис. 2.1: Схема алгоритма быстрой сортировки

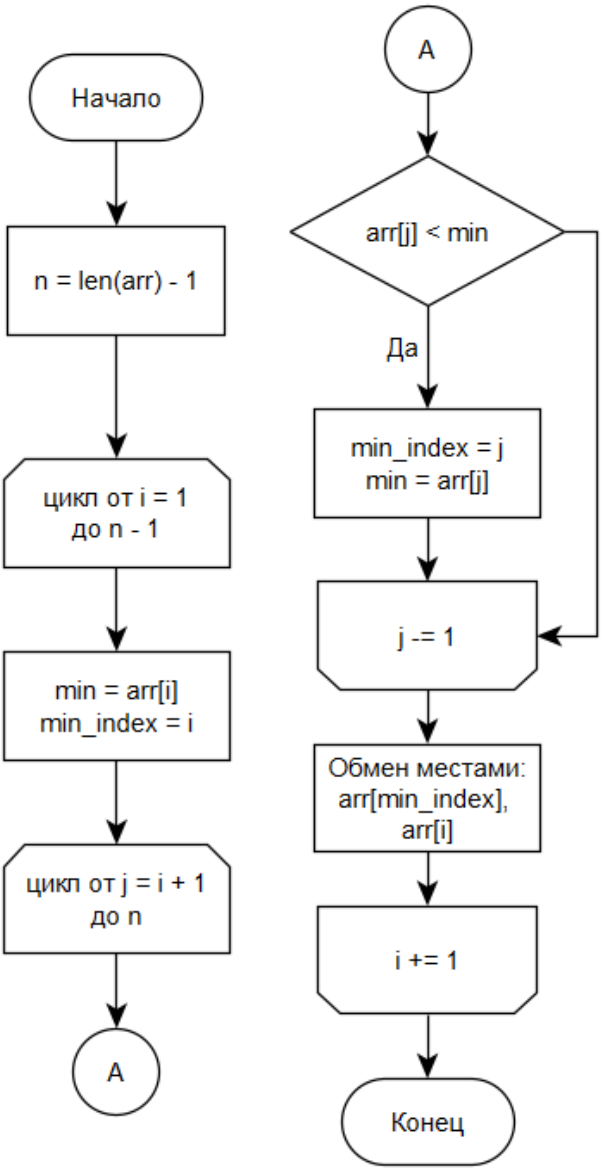

Рис. 2.2: Схема алгоритма сортировки выбором

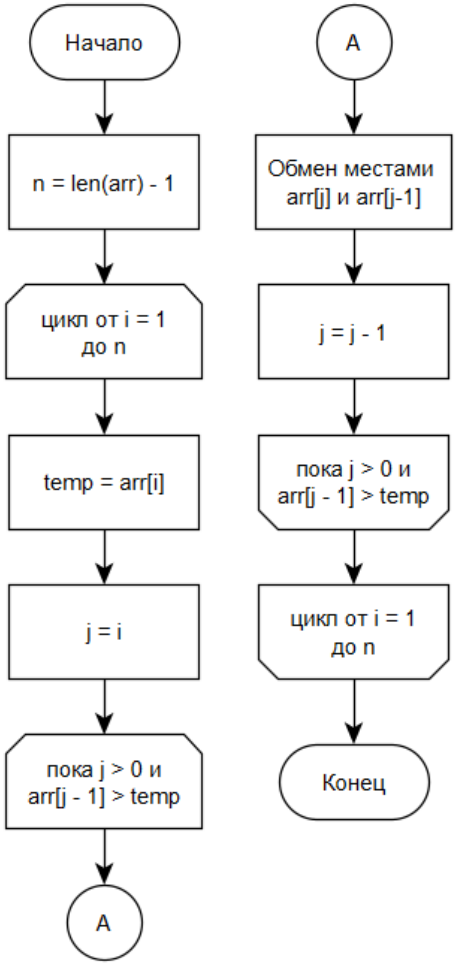

Рис. 2.3: Схема алгоритма сортировки вставками

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

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

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

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

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

3. трудоемкость цикла рассчитывается, как (2.3);

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

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):

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

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

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

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

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

Трудоёмкость в лучшем случае (2.7):

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

Трудоёмкость в худшем случае (2.8):

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

2.3.2 Алгоритм быстрой сортировки

Трудоёмкость в лучшем случае (2.9):

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

Трудоёмкость в худшем случае (2.10):

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

2.3.3 Алгоритм сортировки вставками

Трудоёмкость в лучшем случае (2.11):

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

Трудоёмкость в худшем случае (2.12):

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

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 - C. Данный язык быстр и удобен в использовании. В качестве среды разработки использовался Visual Studio Code.

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.

```
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;
```

```
    }
}
```

Листинг 3.1: Алгоритм сортировки вставками

```
void choice_sort(int *arr, int n)
{
    int min_pos, tmp;
    for (int i = 0; i < n; i++)
    {
        min_pos = i;
        for (int j = i + 1; j < n; j++)
            if (arr[min_pos] > arr[j])
                min_pos = j;

        tmp = arr[min_pos];
        arr[min_pos] = arr[i];
        arr[i] = tmp;
    }
}
```

Листинг 3.2: Алгоритм сортировки выбором

```
void quick_sort(int *array, int left, int right)
{
    int pivot = array[left];
    int l_hold = left;
    int r_hold = right;


    while (left < right)
    {
        while ((array[right] >= pivot) && (left < right))
            right--;

        if (left != right)
        {
            array[left] = array[right];
            left++;
        }

        while ((array[left] <= pivot) && (left < right))
            left++;

        if (left != right)
        {
            array[right] = array[left];
            right--;
        }
```

```
    }

    array[left] = pivot;
    pivot = left;
    left = l_hold;
    right = r_hold;

    if (left < pivot)
        quick_sort(array, left, pivot - 1);
    if (right > pivot)
        quick_sort(array, pivot + 1, right);
}
```

Листинг 3.3: Алгоритм быстрой сортировки

```
double count_time_insertion_sort(int array[], int n)
{
    clock_t start, end;

    start = clock();
    insertion_sort(n, array);
    end = clock();

    return (double)(end - start) / CLOCKS_PER_SEC;
}
```

Листинг 3.4: Пример реализации бенчмарка

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

4.1 Технические характеристики

- Операционная система: Windows 10 64 bit.
- Память: 16 GiB.
- Процессор: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz [2].

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.

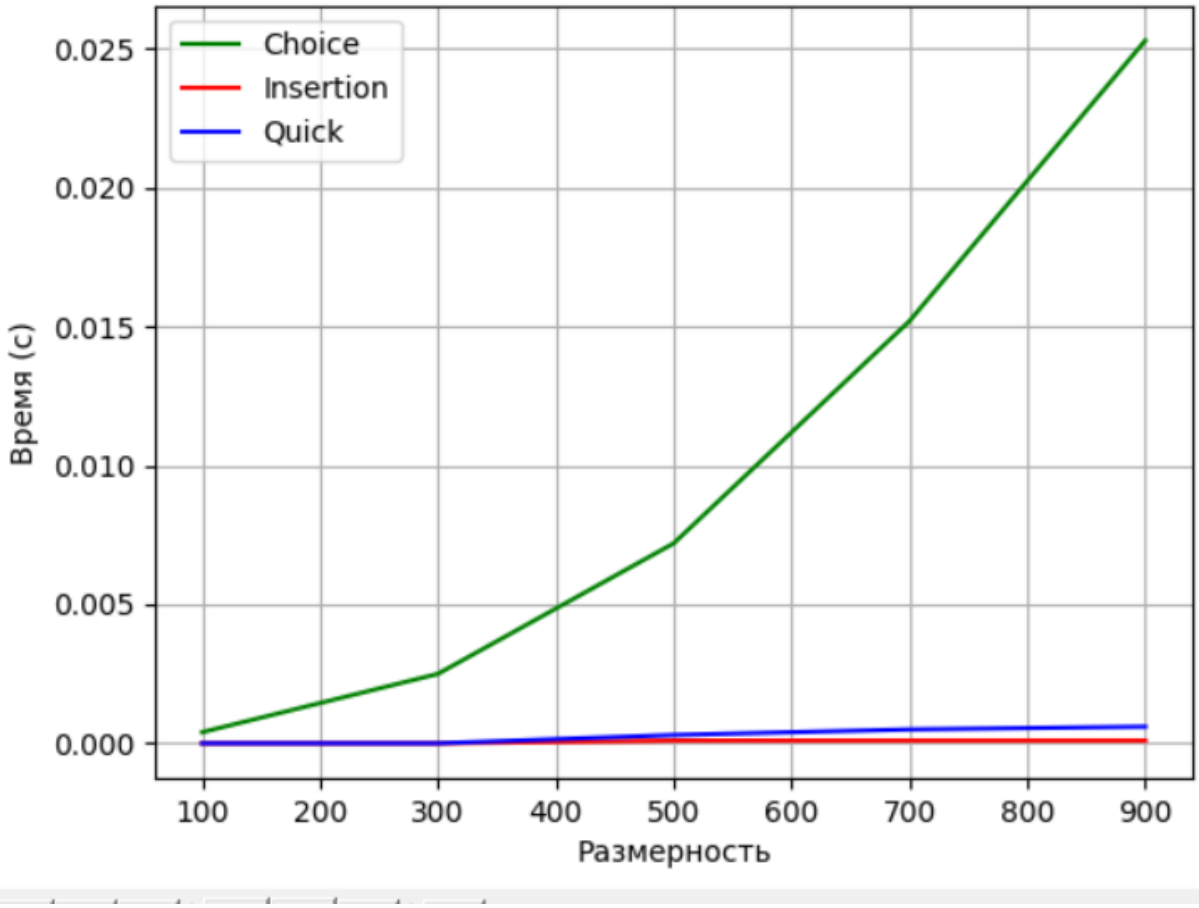

Рис. 4.1: Время работы алгоритмов на лучших данных

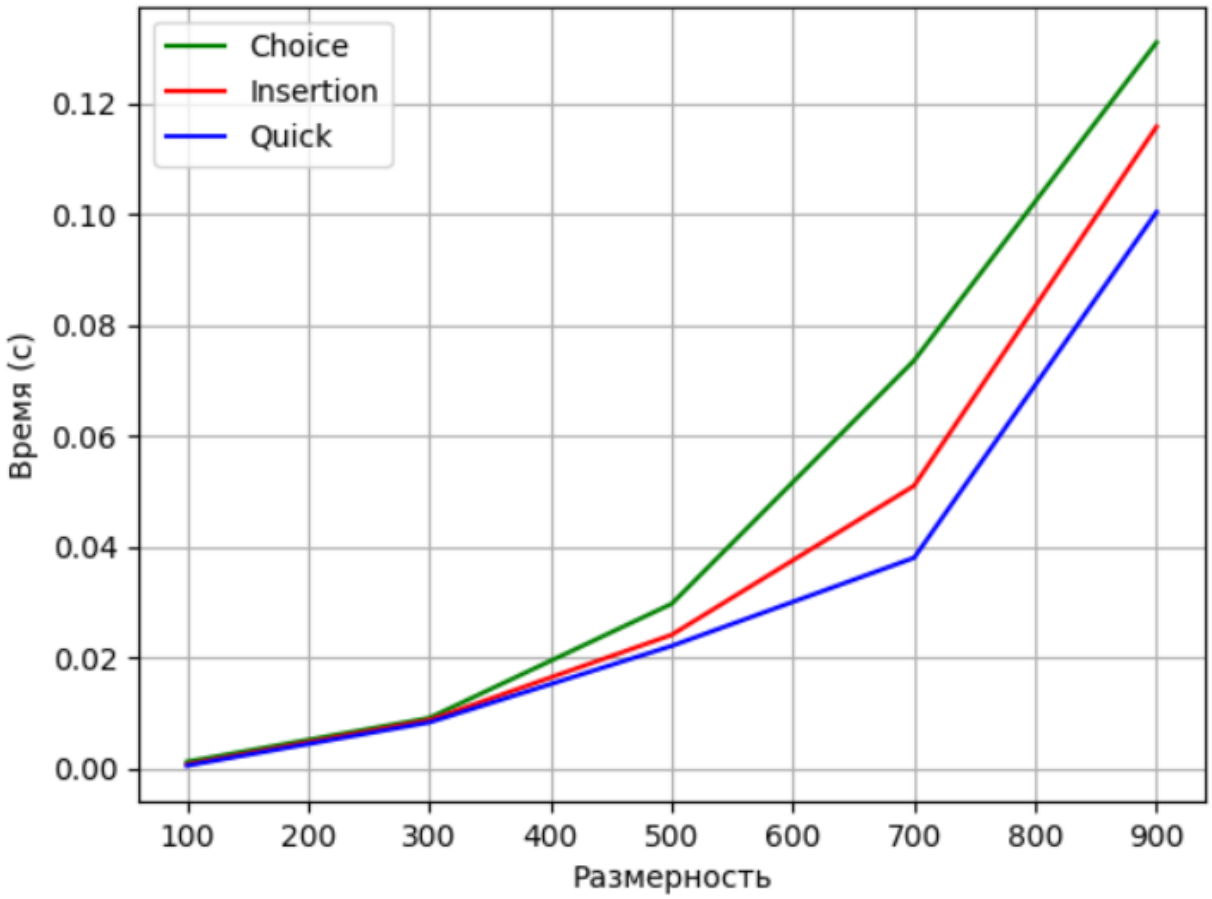

Рис. 4.2: Время работы алгоритмов на худших данных

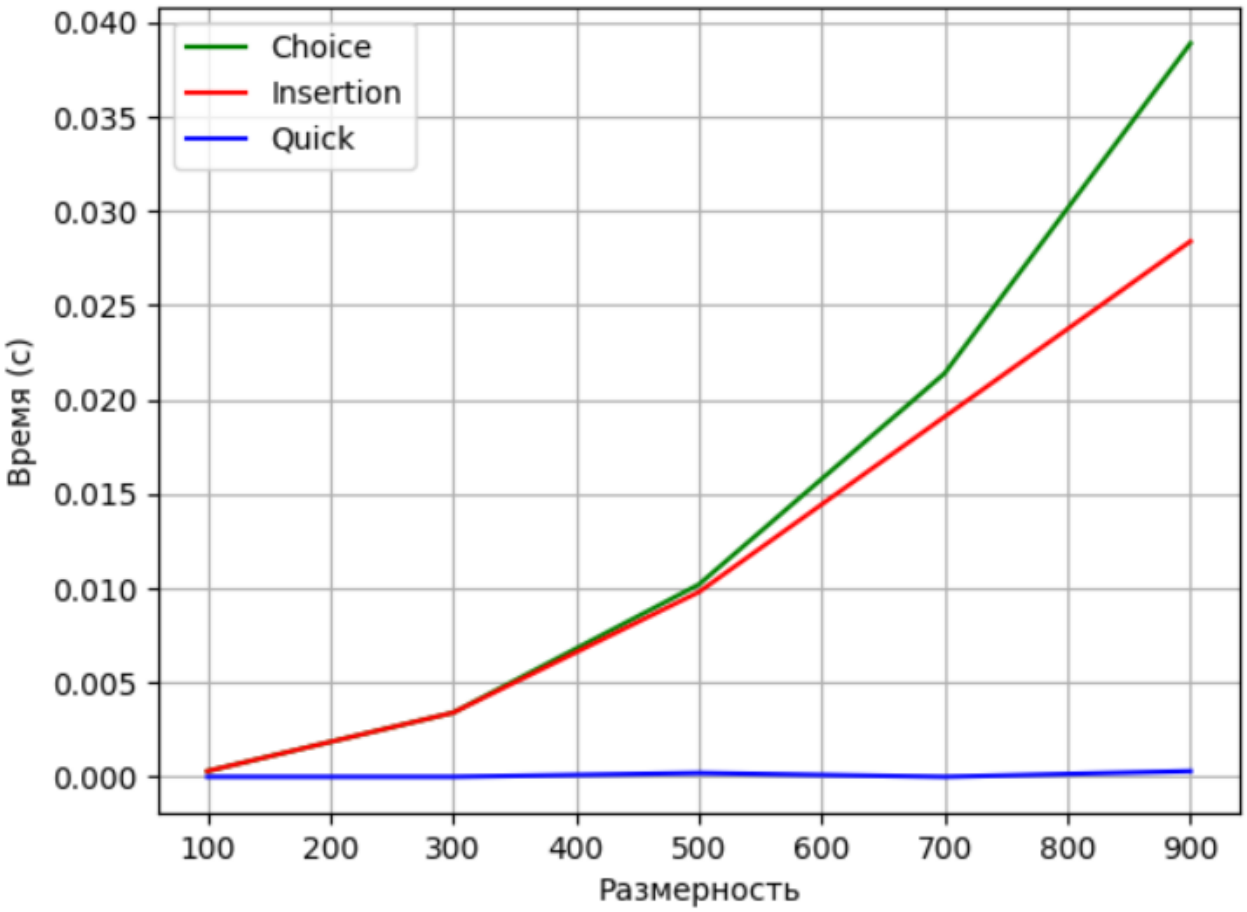

Рис. 4.3: Время работы алгоритмов на рандомных данных

Вывод

Алгоритм быстрой сортировки работает лучше остальных двух в случае с рандомными и худшими данными. С лучшими данными быстрее всего справляется сортировка 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.

Наиболее эффективной оказалась быстрая сортировка. Так же важно отметить, что сортировка выбором работает в худшем случае (случай, когда элементы массива отсортированы в обратном порядке) обгоняет остальные два алгоритма в следствие того, что обмен происходит 1 раз за 1 итерацию внешнего цикла

Литература

[1] Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования.* с. 834.

[2] Процессор Intel® Core™ i7-8550U [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` (дата обращения: 21.09.2020).