

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №3

ПО КУРСУ: "АНАЛИЗ АЛГОРИТМОВ"

## Сортировки

Работу выполнил: Подвашецкий Дмитрий, ИУ7-54Б

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

*Москва, 2019*

# Оглавление

<b>Введение</b>	<b>2</b>
<b>1 Аналитическая часть</b>	<b>3</b>
1.1 Быстрая сортировка . . . . .	3
1.2 Сортировка пузырьком . . . . .	3
1.3 Сортировка вставками . . . . .	3
<b>Вывод</b>	<b>4</b>
<b>2 Конструкторский раздел</b>	<b>5</b>
2.1 Схемы алгоритмов и их анализ . . . . .	5
2.1.1 Быстрая сортировка . . . . .	5
2.1.2 Сортировка пузырьком . . . . .	6
2.1.3 Сортировка вставками . . . . .	7
<b>Вывод</b>	<b>8</b>
<b>3 Технологический раздел</b>	<b>9</b>
3.1 Выбор ЯП . . . . .	9
3.2 Замеры времени . . . . .	9
3.3 Создание псевдослучайного массива . . . . .	9
3.4 Требования к ПО . . . . .	9
3.5 Ленивые вычисления . . . . .	9
3.6 Сведения о модулях программы . . . . .	10
<b>Вывод</b>	<b>11</b>
<b>4 Экспериментальная часть</b>	<b>12</b>
4.1 Сравнительный анализ алгоритмов . . . . .	12
4.1.1 Прямой порядок . . . . .	12
4.1.2 Обратный порядок . . . . .	13
4.1.3 Случайный порядок . . . . .	14
<b>Вывод</b>	<b>15</b>
<b>Заключение</b>	<b>16</b>
<b>Список литературы</b>	<b>17</b>

# Введение

**Алгоритм сортировки** - это алгоритм, позволяющий упорядочить элементы в некотором списке. Сортировки - это основа, которую учат все, кто так или иначе хочет заниматься чем-либо связанным с программированием.

За все время было создано огромное множество различных алгоритмов сортировки, каждая из которых обладает какими-либо особенностями. В данной лабораторной работе я постараюсь это продемонстрировать.

**Задачами** данной лабораторной работы являются:

1. выбор и изучение трех алгоритмов сортировки;
2. реализация выбранных алгоритмов;
3. теоретический анализ сложности;
4. экспериментальное подтверждение различий во временной эффективности алгоритмов;
5. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

# 1 | Аналитическая часть

Для рассмотрения в этой лабораторной работе мною были выбраны алгоритмы:

1. быстрой сортировки;
2. сортировки пузырьком;
3. сортировки вставками.

## 1.1 Быстрая сортировка

Суть данного алгоритма заключается в выборе некоторого опорного элемента (обычно выбирают либо последний, либо средний) и дальнейшем разбиении списка на два подсписка: все элементы меньше опорного и все те, что больше опорного. Далее для каждого из двух подписков рекурсивно применяется тот же алгоритм сортировки.

Обозначим:

$qSort(list)$  - применение алгоритма быстрой сортировки к некоторому списку  $list$ .

$$list = l_0, l_1, \dots, l_n$$

$$listL = l_i : l_i \leq l_0, i = 1..n$$

$$listR = l_i : l_i > l_0, i = 1..n$$

Тогда алгоритм быстрой сортировки можно записать как:

$$qSort(list) = qSort(listL) + l_0 + qSort(listR) \quad (1.1)$$

## 1.2 Сортировка пузырьком

Данный алгоритм заключается в проходе списка слева направо до конца. Если текущий элемент больше следующего, то необходимо поменять их местами (для сортировки по возрастанию). Заметим, что после, этот процесс необходимо повторять до тех пор, пока массив не будет отсортирован. В случае если алгоритм никак не модифицирован, то необходимо повторить кол-во раз, равного длине массива.

## 1.3 Сортировка вставками

Суть этого алгоритма заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем.

Стоит отметить что массив из 1-го элемента считается отсортированным. [1] В ходе работы данного алгоритма, при обработке  $i$ -го элемента можно быть уверенным в том, что левая часть является полностью отсортированной.

## Вывод

В данном разделе мною было рассмотрены и вкратце описаны рассматриваемые мною алгоритмы.

## 2 | Конструкторский раздел

### 2.1 Схемы алгоритмов и их анализ

#### 2.1.1 Быстрая сортировка

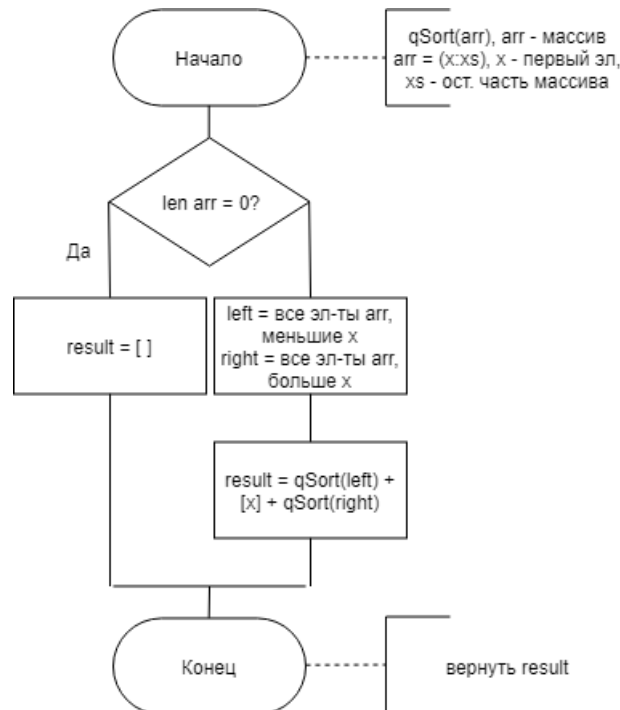


Рис 2. Схема алгоритма быстрой сортировки.

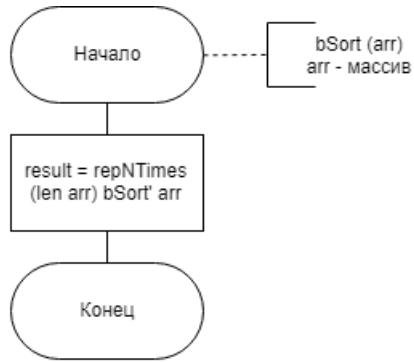
#### Лучший случай:

Лучший случай достигается тогда, когда при каждом вызове функции массив делится пополам ( $\pm 1$  элемент). Рекурсия завершается если в обрабатываемом массиве остался только 1 элемент. В таком случае, массив делится на  $N/2$  и  $N/2$ , следовательно чтобы найти максимальную глубину рекурсии необходимо найти  $i$  из уравнения  $N/2^i = 1$ .  $i = \log_2(N)$ , из чего следует, что трудоемкость алгоритма равна:  $N * \log_2(N)$

#### Худший случай:

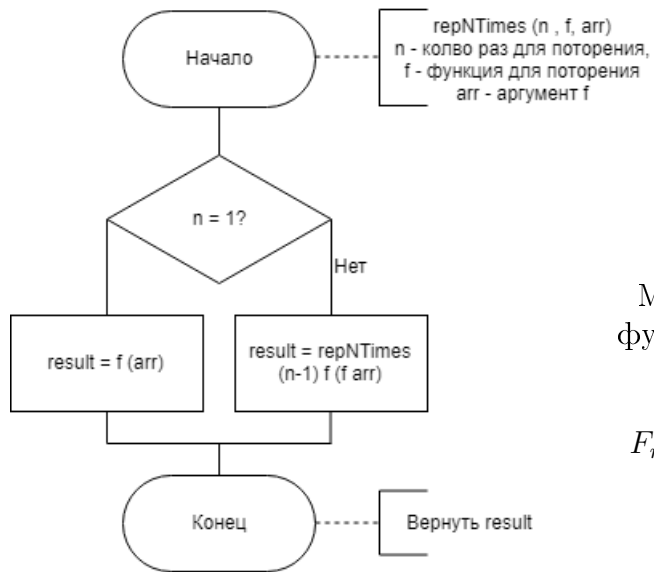
В худшем случае каждое разделение дает два подмассива длины 1 и  $N-1$ . При выборе опорного элемента первым или последним, такой эффект даст полностью отсортированный массив. Максимальная глубина рекурсии в таком случае равна  $N$ , из чего следует, что трудоемкость алгоритма равно:  $O(N^2)$

## 2.1.2 Сортировка пузырьком



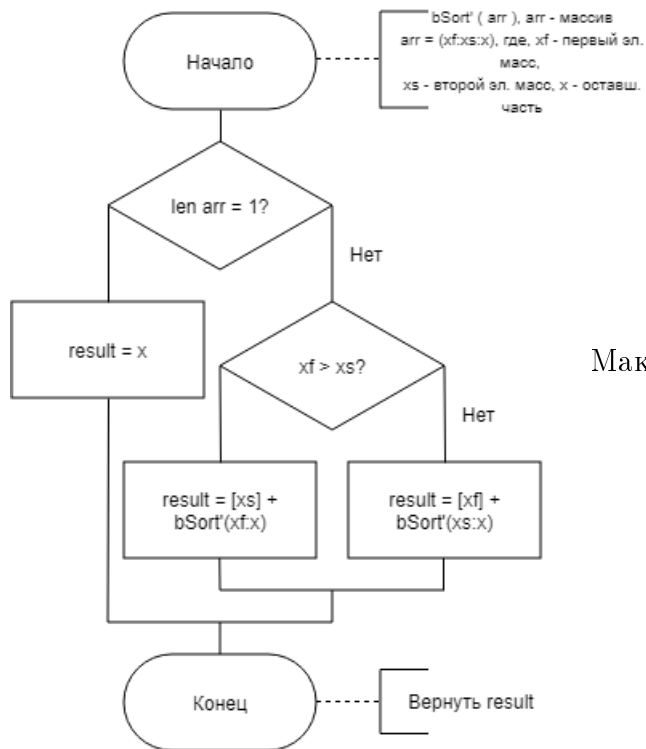
bSort - это функция 'надстройка',  
единственная её задача - вызов  
функции repNTimes, следовательно

$$F_{bSort} = F_{repNTimes} \quad (2.1)$$



Максимальная глубина рекурсии  
функции repNTimes = N, что равно  
длине массива

$$F_{repNTimes} = N(2 + F_{bSort'}) - 1 \quad (2.2)$$



Максимальная глубина рекурсии bSort' так же равна N

$$F_{bSort'} = 3N - 1 \quad (2.3)$$

Рис 2. Схема алгоритма сортировки пузырьком.

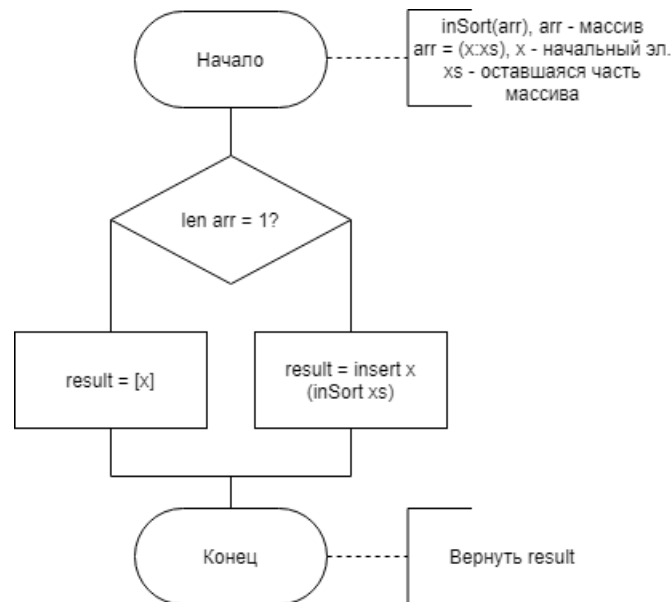
#### Анализ трудоемкости:

$$F_{bSort} = F_{repNTimes} = N(2 + F_{bSort'}) - 1 = N(2 + 3N - 1) - 1 = N + 3N^2 - 1 \quad (2.4)$$

Для данной реализации алгоритма сортировки пузырьком, нет различий в трудоемкости при обработке обратно отсортированного массива, прямого или случайного.

Общая сложность алгоритма для всех случаев:  $O(N^2)$

### 2.1.3 Сортировка вставками





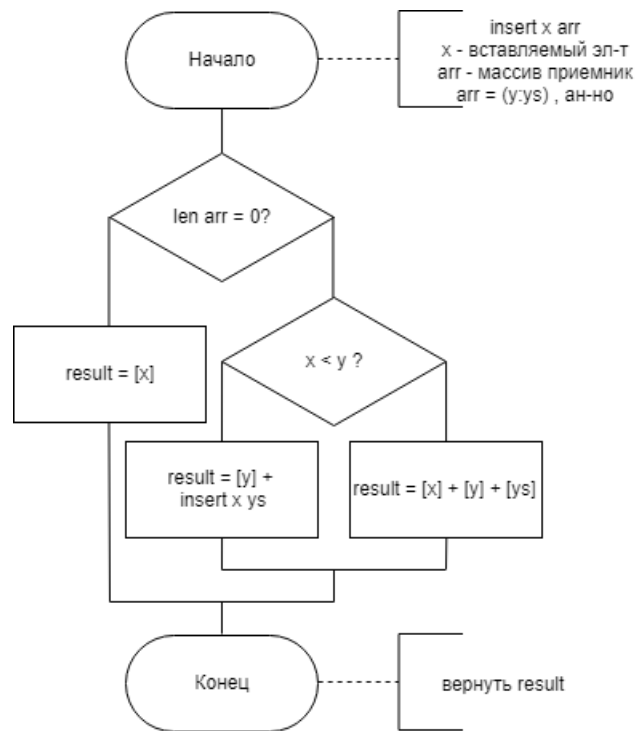


Рис 3. Схема алгоритма сортировки вставками.

### Анализ трудоемкости:

Согласно [3]:

**Лучший случай** -  $O(N)$

**Худший случай** -  $O(N^2)$

## Вывод

Таким образом, в данном разделе были рассмотрены исследуемые алгоритмы, а так же вычислены их трудоемкости.

## 3 | Технологический раздел

### 3.1 Выбор ЯП

В качестве языка программирования был выбран Haskell, для ознакомления с ним.

### 3.2 Замеры времени

Замер времени работы алгоритмов производился при помощи функций `getCurrentTime`, `diffUTCTime` из библиотеки `Data.Time.Clock`. [2]

Также производится усреднение времени работы алгоритмов. Для этого время считается для 10 вызовов, и после делится на 10.

### 3.3 Создание псевдослучайного массива

Для того, чтобы создать массив с минимальным кол-вом отсортированных под-массивов я завел дополнительный массив:  $A = [3, 5, 7, 11, 13, 17, 23]$  И вычислял  $i$ -тый эл-т массива как  $Arr_i = [i \bmod (a[i \bmod 7])]$

Таким образом каждый раз создается один и тот же псевдослучайны массив. (Это было сделано, так как я не хотел разбираться с библиотекой случайных чисел)

### 3.4 Требования к ПО

**Требования к вводу:**

1. Массив целых чисел

**Требования к программе:**

1. Корректный ввод, корректный вывод, программа не должна аварийно завершаться

### 3.5 Ленивые вычисления

Для того, чтобы избавиться от ленивых вычисления использовались Bang patterns. Для этого программа компилировалась с ключем `-XBangPatterns`[4].

## 3.6 Сведения о модулях программы

Программа состоит из:

- main.hs - главный файл программы
- bsort.hs - файл с реализацией алгоритма сортировки пузырьком (Листинг 3.1.)
- inSort.hs - файл с реализацией алгоритма сортировки вставками (Листинг 3.2.)
- qsort.hs - файл с реализацией алгоритма быстрой сортировки (Листинг 3.3.)

Листинг 3.1: Сортировка пузырьком

```
1  bSort' :: (Ord a) => [a] -> [a]
2  bSort' [] = []
3  bSort' [x] = [x]
4  bSort' (xf:xs:x) =
5    if xf > xs
6    then (xs:bSort' (xf:x))
7    else (xf:bSort' (xs:x))
8
9  repNTimes :: (Ord a) => Int -> ([a] -> [a]) -> [a] -> [a]
10 repNTimes 1 f x = f x
11 repNTimes n f x = repNTimes (n-1) f (f x)
12
13 bSort :: (Ord a) => [a] -> [a]
14 bSort x = repNTimes (length x) bSort' x
```

Листинг 3.2: Сортировка вставками

```
1  insert :: (Ord a) => a -> [a] -> [a]
2  insert x [] = [x]
3  insert x (y:ys) = if x < y
4    then x:y:ys
5    else y : insert x ys
6
7  inSort :: (Ord a) => [a] -> [a]
8  inSort [x] = [x]
9  inSort (x:xs) = insert x (inSort xs)
```

Листинг 3.3: Быстрая сортировка

```
1  qSort :: (Ord a) => [a] -> [a]
2  qSort [] = []
3  qSort (x:xs) = qSort(left) ++ [x] ++ qSort(right)
4  where
5    left = [a | a <- xs, a <= x]
6    right = [a | a <- xs, a > x]
```

## Вывод

В данном разделе мною были реализованы алгоритмы рассматриваемых сортировок.

## 4 | Экспериментальная часть

### 4.1 Сравнительный анализ алгоритмов

#### 4.1.1 Прямой порядок

Анализируя Таблицу. 1. можно увидеть, что сортировка Пузырьком работает примерно в 120 раз медленнее чем сортировка Вставками, и в 75 медленнее чем быстрая сортировка.

Таблица. 1. Сравнение алгоритмов при прямо отсортированном массиве.

Размер	bSort (с)	inSort (с)	qSort (с)
1000	0.01922	2e-05	0.00026
1100	0.02103	0.00019	0.00033
1200	0.02419	7e-05	0.00028
1300	0.02832	9e-05	0.00024
1400	0.03067	0.00023	0.00031
1500	0.03516	6e-05	0.00034
1600	0.03759	5e-05	0.00031
1700	0.0454	0.0001	0.00033
1800	0.05219	0.0002	0.00034
1900	0.06051	6e-05	0.00033
2000	0.06591	9e-05	0.00037

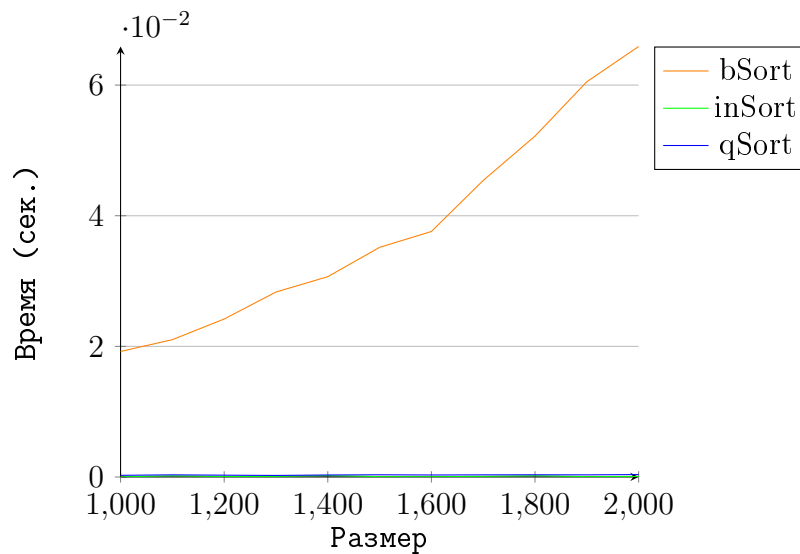


Рис. 1. График сравнения алгоритмов сортировки при прямо отсортированном массиве. (Таблица. 1.)

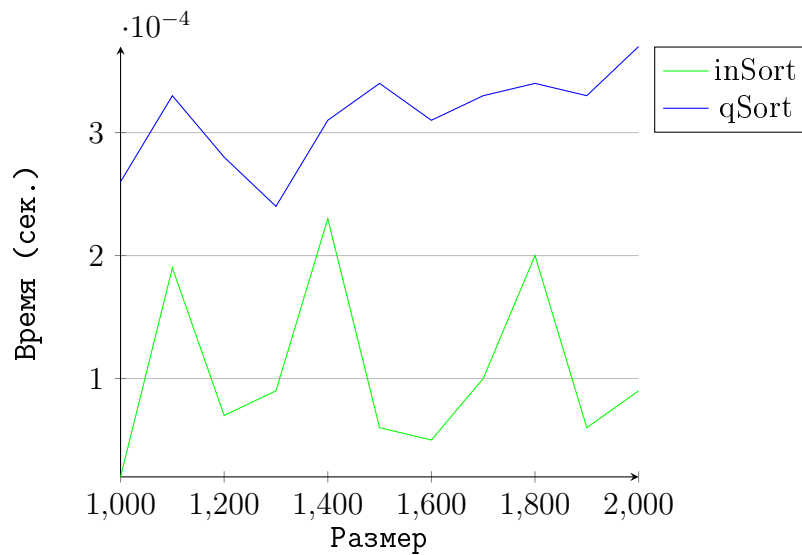


Рис. 2. График сравнения алгоритмов сортировки при прямо отсортированном массиве без сортировки пузырьком. (Таблица. 1.)

#### 4.1.2 Обратный порядок

По данным из Таблицы 2. можно увидеть, что сортировка Пузырьком работает в 150 раз медленнее чем Вставками и в 2 раза быстрее чем Быстрая. На данном массиве данных, Быстрая сортировка показывает наихудший результат. Результат сортировки Пузырьком не изменился.

Таблица. 2. Сравнение алгоритмов при обратно отсортированном массиве.

Размер	bSort (c)	inSort (c)	qSort (c)
1000	0.02464	8e-05	0.05463
1100	0.02384	4e-05	0.05893
1200	0.03504	9e-05	0.08216
1300	0.03601	5e-05	0.09325
1400	0.04114	9e-05	0.10566
1500	0.04539	0.00029	0.11752
1600	0.05999	0.00012	0.14982
1700	0.07741	0.00016	0.18477
1800	0.07027	9e-05	0.17601
1900	0.08	0.00016	0.19965
2000	0.08919	0.00015	0.22478

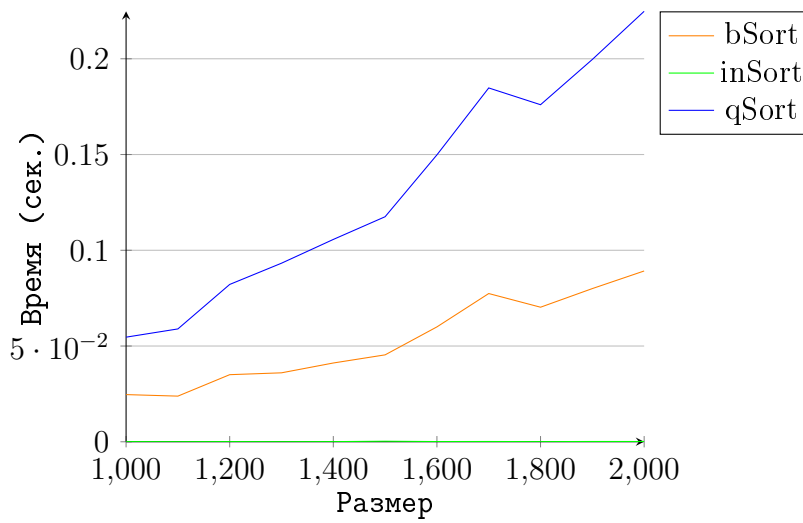


Рис. 3. График сравнения алгоритмов сортировки при обратном отсортированном массиве. (Таблица. 1.)

Из Рис. 3. видно, что Быстрая сортировка показывает наихудший результат. Это связано с тем, что в результате разбиения массива из  $N$  элементов появляется на подмассива: длины 1 и  $(N-1)$ . Массив длины 1 обрабатываться не будет, а массив длины  $(N-1)$  разобьется на подмассивы 1 и  $(N-2)$ . Можно заметить, что в итоге будет обратно  $N \cdot (N-1) \cdot (N-2) \cdot \dots \cdot 1$  элементов, что равно  $N!$ . Это объясняет почему скорость Быстрой сортировки наименьшая.

Такие же рассуждения можно применить и к массиву прямой отсортированности. Но, как следует из Рис. 1., скорость Быстрой сортировки в таком случае не совпадет со случаем обратного отсортированного массива. Вероятнее всего это связано с некоторыми особенностями языка Haskell, к сожалению не имеется возможности предоставить точную информацию по данному поводу.

### 4.1.3 Случайный порядок

Результаты для массива со случайным порядком практически не отличаются от массивов с прямым порядком, результаты которого представлены выше.

Таблица. 3. Сравнение алгоритмов при случайном массиве.

Размер	bSort (c)	inSort (c)	qSort (c)
1000	0.01937	0.00029	0.00065
1100	0.01962	0.00019	0.00064
1200	0.02371	0.00026	0.00042
1300	0.02616	0.00015	0.00067
1400	0.02823	0.00033	0.00068
1500	0.03296	0.00019	0.00095
1600	0.03725	0.00025	0.00071
1700	0.04761	0.00042	0.00143
1800	0.05449	0.00035	0.00133
1900	0.06122	0.00028	0.00145
2000	0.06756	0.00029	0.0022

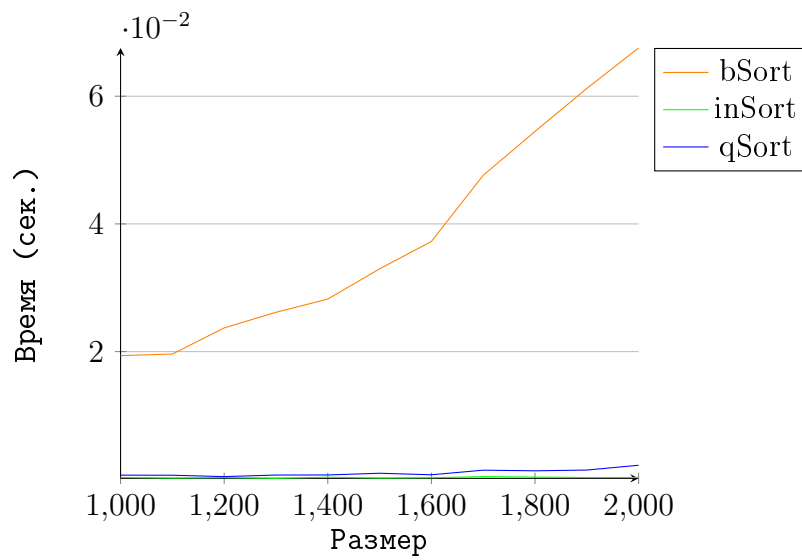


Рис. 4. График сравнения алгоритмов сортировки при случайном массиве. (Таблица. 1.)

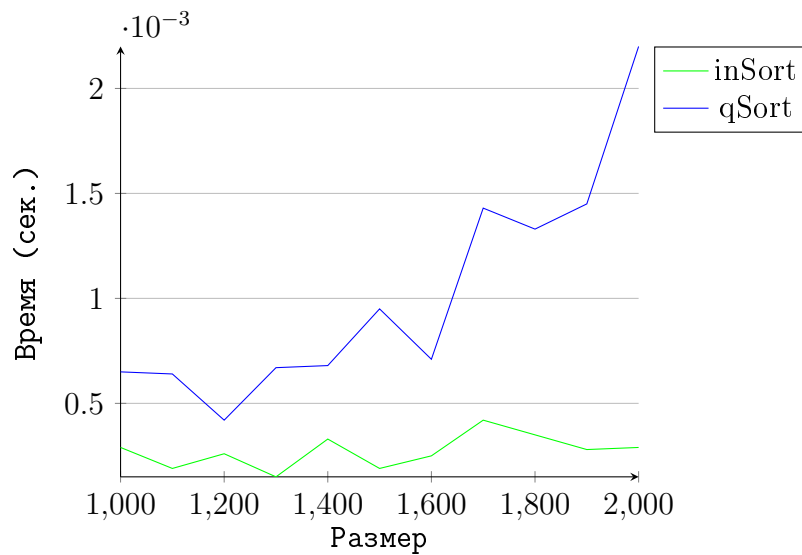


Рис. 5. График сравнения алгоритмов сортировки при случайном массиве без сортировки вставками. (Таблица. 1.)

## Вывод

В данном разделе были произведены оценки времени работы рассматриваемых алгоритмов. Из этого можно сделать вывод, что сортировка вставками, магическим образом, во всех случаях выигрывает у Пузырьковой и Быстрой. (Я не знаю почему :( )



# Заключение

При написании данной лабораторной работы, мною были изучены и реализованы алгоритмы Быстрой, Пузырьковой сортировки и сортировки Вставками.

Так же были теоретически проанализированы сложности этих алгоритмов.

Практически были подтверждены различия между этими сортировками. В результате тестов, было обнаружено, что сортировка Вставками работает аномально быстро. Вероятно это связано с особенностями языка Haskell.

В итоге можно сделать вывод, что, почему-то, в любом случае предпочтительнее использовать сортировку Вставками.

# Список литературы

1. В мире алгоритмов: Сортировка Вставками. [Электронный ресурс] Режим доступа: <https://habr.com/ru/post/181271/> Последняя дата обращения: 12.11.2019
2. Быстрая сортировка :: Quick sort. [Электронный ресурс] Режим доступа: <http://algotab.valemak.com/quick> Последняя дата обращения: 24.11.2019
3. Сортировка простыми вставками :: Insertion sort. [Электронный ресурс] Режим доступа: <http://algotab.valemak.com/insertion-simple> Последняя дата обращения: 24.11.2019
4. Bang patterns :: Insertion sort. [Электронный ресурс] Режим доступа: [https://downloads.haskell.org/~ghc/7.8.4/docs/html/users\\_guide/bang-patterns.html](https://downloads.haskell.org/~ghc/7.8.4/docs/html/users_guide/bang-patterns.html) Последняя дата обращения: 26.12.2019