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»

О Т Ч Е Т

по лабораторной работе № 0 3

Название ***Сортировки***

Дисциплина: ***Анализ алгоритмов***

Студент	***ИУ7И-56Б***		**Нгуен Ф. С.**
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			**Волокова Л. Л.**
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Оглавление

Введение

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

В настоящее время в любом программном проекте необходимо обрабатывать большое число однотипных данных. Такие данные удобно обрабатывать, используя массив - именованную последовательность однотипных данных. Благодаря тому, что элементы массива расположены последовательно, упрощается их обработка.

Очень часто требуется сортировать данные по какому-либо признаку, или ключу. В отсортированных массивах значительно быстрее можно выполнять поиск определённых данных по ключу. Существует множество алгоритмов сортировки массивов, которые применяются при различных условиях в зависимости от задач, стоящих перед программой. В данной лабораторной работе изучаются некоторые из алгоритмов сортировки массивов на предмет их эффективности.

Цель лабораторной работы - изучение трех алгоритмов сортировки массивов.

Для того чтобы добиться этой цели, были поставлены следующие задачи:

- изучить и реализовать алгоритмы сортировок массивов: сортировка вставками, пузырьком, подсчётом;
- оценить трудоёмкости алгоритмов;
- выполнить сравнительный анализ алгоритмов на основе трудоёмкости и используемой памяти;
- оценить и сравнить эффективности алгоритмов по времени.

I. Аналитическая часть

1. Описание алгоритмов

i. *Сортировка пузырьком*

Сравнить каждую пару элементов, если они не в правильном порядке, поменять их местами

ii. 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.

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.

iii. Сортировка подсчётом

Сортировка подсчётом— алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000.

Предположим, что входной массив состоит из n целых чисел в диапазоне от 0 до k-1. Помимо входного массива A потребуется два вспомогательных массива — C[0..k - 1] для счётчика и B[0..n - 1] для отсортированного массива. Последовательно пройдём по массиву A и запишем в C[i] количество чисел, равных i. Теперь достаточно пройти по массиву C и для каждого числа number из диапазона допустимых значений последовательно записать в массив A число number C[number] раз.

2. Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

- +, -, /, \%, ==, !=, <, >, <=, >=, [], ++, \-- -- имеют трудоемкость 1
- трудоемкость оператора выбора

  if (условие) then

  A

  Else

  B

  рассчитывается, как

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

- трудоемкость цикла рассчитывается, как

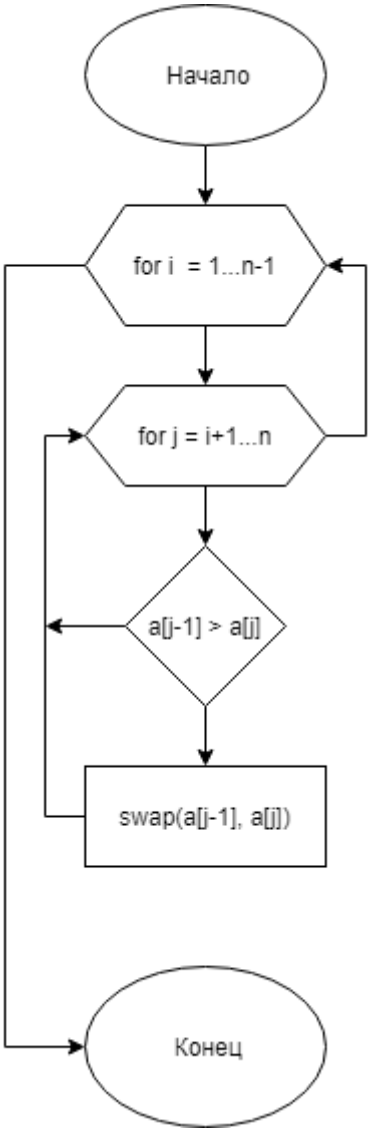
$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$$

- трудоемкость вызова метода равна 0
- трудоемкость вызова функции равна 1

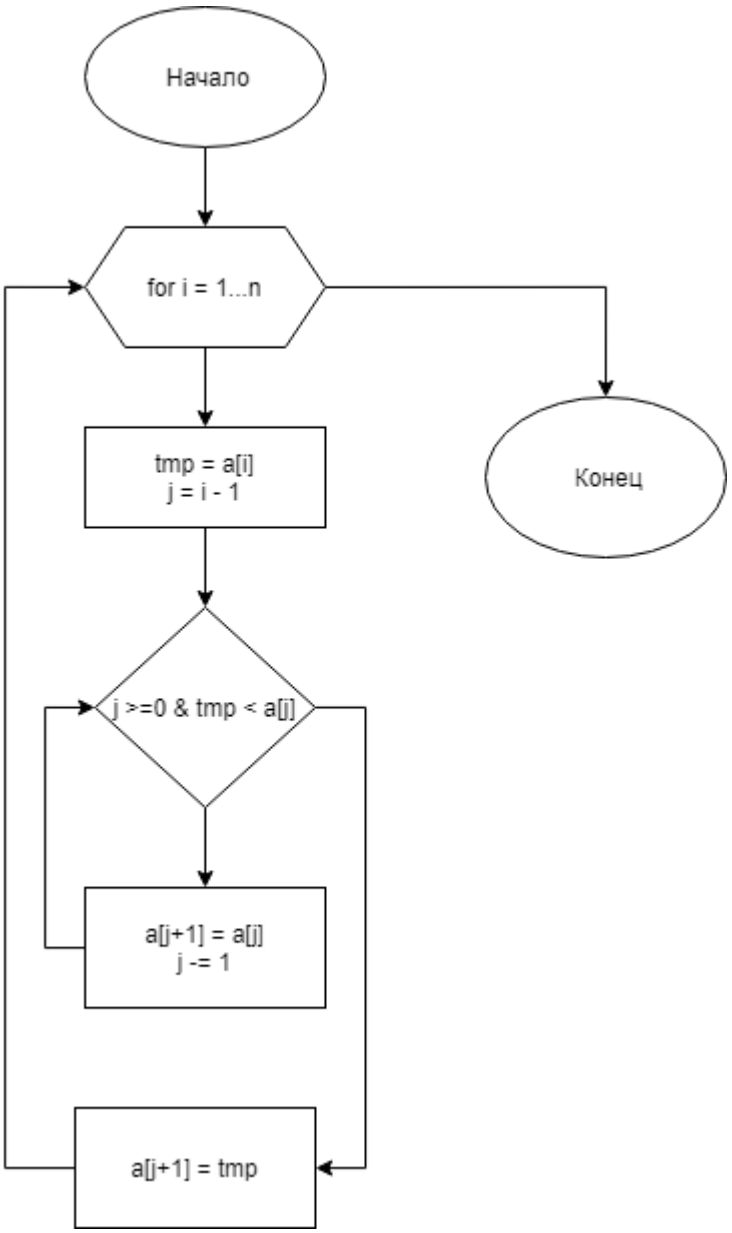
II. Конструкторская часть

1. Схемы алгоритмов

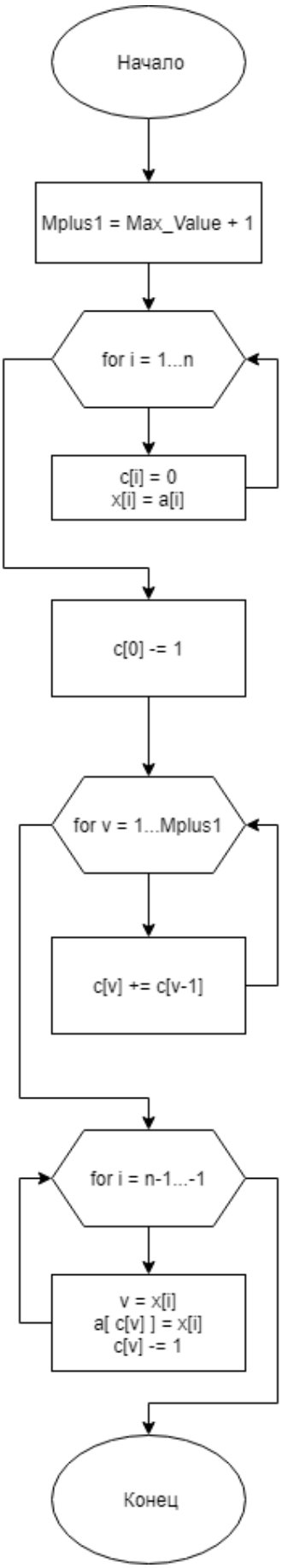
Сортировка пузырьком



Сортировка вставками



Сортировка подсчётом



2. Оценка трудоёмкости

Пусть дан массив A длиной N. Рассмотрим трудоемкость трёх алгоритмов сортировки.

i. *Сортировка пузырьком:*

$$f_{1\text{ сравнение}} = 4 + 2 + \begin{cases} 0, \text{Лучший случай} \\ 9, \text{Худший случай} \end{cases}$$

$$Q_{\text{сравнение}} = \frac{(N-1)N}{2}$$

$$f_{\text{случаев}} = 2 + (N-1)(2+2) = 4N - 2$$

Общая трудоемкость алгоритма:

$$f = f_{\text{случаев}} + Q_{\text{сравнение}} \cdot f_{1\text{ сравнение}} = \begin{cases} 3N^2 + N - 2, & \text{Лучший случай} \\ \frac{15}{2}N^2 - \frac{7N}{2} - 2, & \text{Худший случай} \end{cases}$$

Трудоемкость: O(N^2)

- Лучший случай - массив уже отсортирован
- Худший случай - массив отсортирован в обратном

ii. *Сортировка вставками:*

$$f_{for} = 2 + N(9+4) = 13N + 2$$

$$f_{1\,while} = 4 + 4 + 1 = 9$$

$$Q_{while} = \begin{cases} 0, & \text{Лучший случай} \\ \frac{N(N-1)}{2}, & \text{Худший случай} \end{cases}$$

Общая трудоемкость алгоритма:

$$f = f_{for} + f_{1\,while} * Q_{while} = \begin{cases} 13N + 2, \text{Лучший случай} \\ \frac{9N^2}{2} + \frac{15N}{2} + 2, \text{Худший случай} \end{cases}$$

Трудоемкость: O(N^2)

- Лучший случай - массив уже отсортирован
- Худший случай - массив отсортирован в обратном

iii. Сортировка подсчётом

$f = 2 + [2 + (M + 1)(2 + 2)] + [2 + N(2 + 3)] + 2 + [2 + (M + 1)(2 + 4)] + [3 + N(2 + 8)] = 15N + 10M + 23$

Трудоемкость: O(max(M, N))

3. Замер используемой памяти

В каждом из алгоритмов требуется хранить исходный массив A длиной N. Таким образом, под хранение массива требуется 4*N байт памяти.

Однако в алгоритме Сортировки подсчётом потребуется два вспомогательных массива — C[0..M] для счётчика и B[0..N - 1] для отсортированного массива, требуется (M+N)*4 байт памяти

III. Технологическая часть

1. Требования к программному обеспечению:

На вход подаются размер массива и сам массив. На выход программа выдаёт три массива, которые являются результатами работы трёх различных алгоритмов сортировки. Сортировка выполняется по возрастанию.

2. Средства реализации

Для реализации программы был использован язык Python. Для замера процессорного времени была использована функция time() из библиотеки time.

3. Реализации алгоритмов

i. *Сортировка пузырьком*

```
1.  def bubbleSort(a):
2.      n = len(a)
3.      nmin1 = n - 1
4.      for i in range(nmin1):
5.          for j in range(nmin1, i, -1):
6.              if (a[j-1] > a[j]):
7.                  tmp = a[j]
8.                  a[j] = a[j - 1]
9.                  a[j - 1] = tmp
```

ii. *Сортировка вставками*

```
1.   def insertionSort(a):
2.      n = len(a)
3.      for i in range(1, n):
4.          tmp = a[i]
5.          j = i - 1
6.          while ( j >= 0) and (tmp < a[j]):
7.              a[j+1] = a[j]
8.              j -= 1
9.          a[j + 1] = tmp
```

iii. *Сортировка подсчётом*

```
1.  def distributionCounting(a):
2.      M = Max_Value
3.      Mplus1 = M + 1
4.      n = len(a)
5.      c = [0 for i in range(Mplus1)]
6.      x = a.copy()
7.
8.      for i in range(n):
9.          c[a[i]] += 1
10.
11.     c[0] -= 1
12.
13.     for v in range(1, Mplus1):
14.         c[v] += c[v-1]
15.
```

```
    for i in range(n-1, -1, -1):
        v = x[i]
        a[c[v]] = x[i]
        c[v] -= 1
```

4. Тесты

	Массив	Ожидание
1	1, 2, 3, 4, 5, 6	1, 2, 3, 4, 5, 6
2	9, 8, 7, 6, 5, 4, 3	3, 4, 5, 6, 7, 8, 9
3	9, 7, 5, 3, 8, 6, 4	3, 4, 5, 6, 7, 8, 9
4	4, 4, 4, 4, 4, 4, 4, 4	4, 4, 4, 4, 4, 4, 4, 4
5	4, 7, 3, 7, 2, 5, 2, 9	2, 2, 3, 4, 5, 7, 7, 9
6	1, 3, 5, 7, 9, 2, 4, 6, 8	1, 2, 3, 4, 5, 6, 7, 8, 9
7	6, 5, 4, 3, 2, 9	2, 3, 4, 5, 6, 9
8	5	5
9	6, 5	5, 6

IV. Экспериментальная часть

1. Примеры работы

```
N =  6
Array: 6  5  4  3  2  9
Bubble Sort   : [2, 3, 4, 5, 6, 9]
Insertion Sort   : [2, 3, 4, 5, 6, 9]
Distric Counting   : [2, 3, 4, 5, 6, 9]
>>>
```

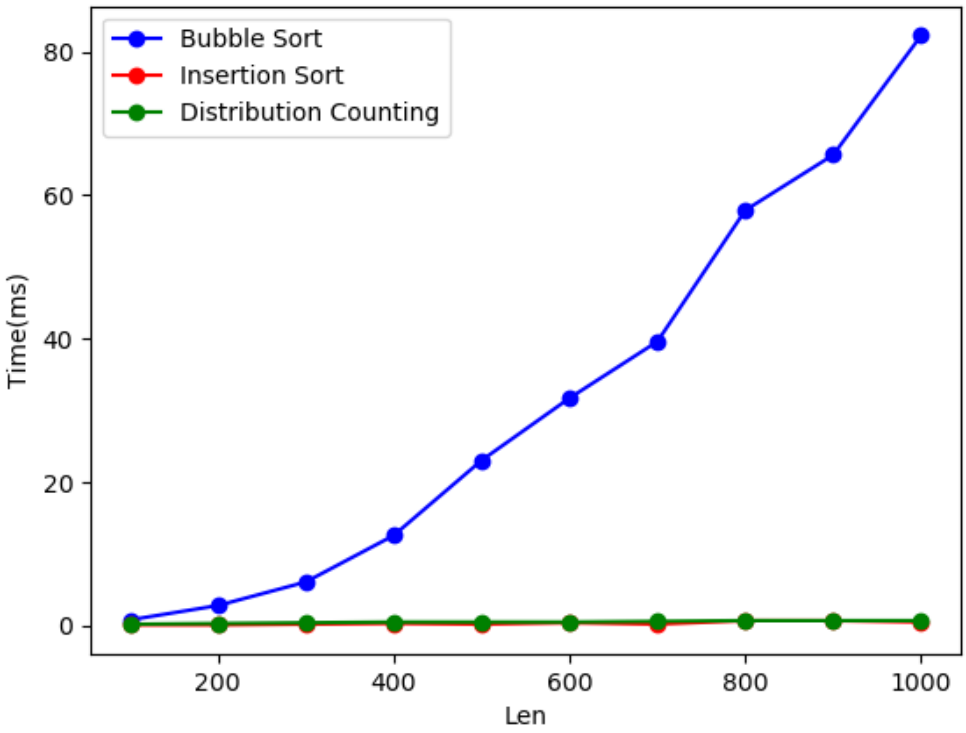
2. Сравнение работы алгоритмов

i. Лучшее время

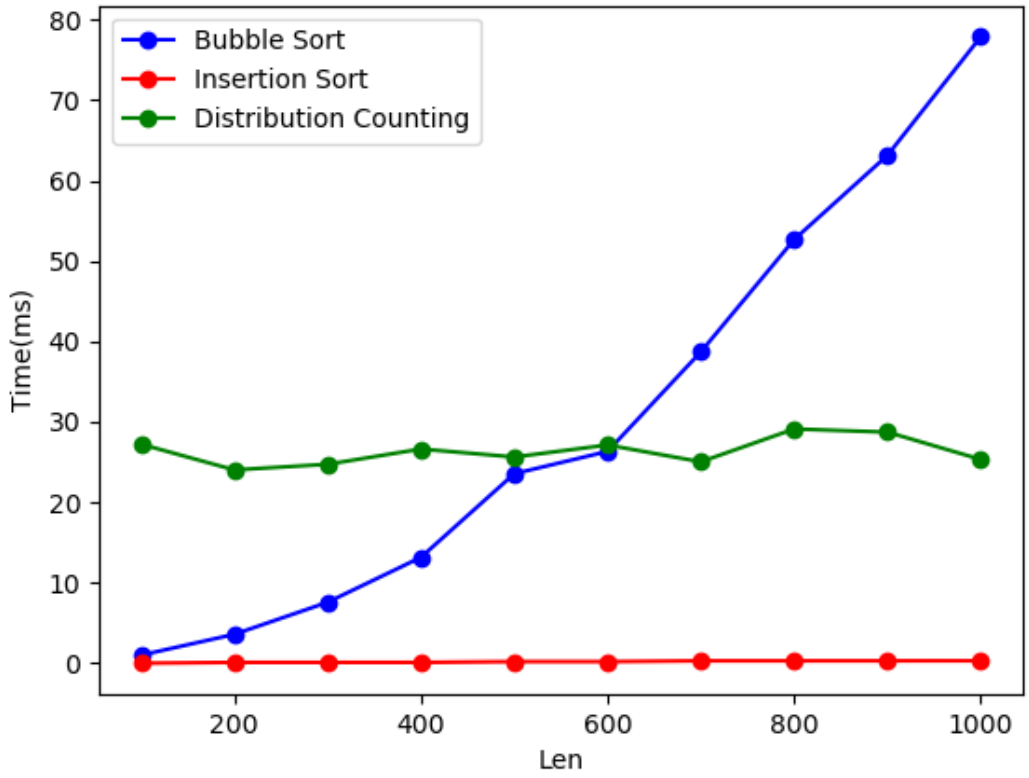
Для сравнения лучшего времени работы алгоритмов сортировки массивов были использованы отсортированные массивы длиной от 100 до 1000 с шагом 100. Значение массива от 0 до 1000 и от 0 до 100 000.

Эксперимент для более точного результата повторялся 10 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице и на рисунке.

	Max_Value = 1000			Max_Value = 100 000		
	Сортировка пузырьком	**Сортировка вставками**	**Сортировка подсчётом**	**Сортировка пузырьком**	**Сортировка вставками**	**Сортировка подсчётом**
100	0.79	0.09	0.19	1.00	0.02	27.03
200	2.75	0.09	0.29	3.59	0.09	24.03
300	6.08	0.19	0.40	7.57	0.1	24.73
400	12.57	0.29	0.48	13.16	0.1	26.62
500	23.03	0.20	0.49	23.55	0.19	25.63
600	31.71	0.39	0.50	26.33	0.20	27.13
700	39.61	0.45	0.59	38.71	0.29	25.03
800	57.91	0.70	0.70	52.67	0.29	29.02
900	65.62	0.69	0.69	63.14	0.29	28.72
1000	82.27	0.73	0.754	77.88	0.30	25.33



Значение массива от 0 до 1000



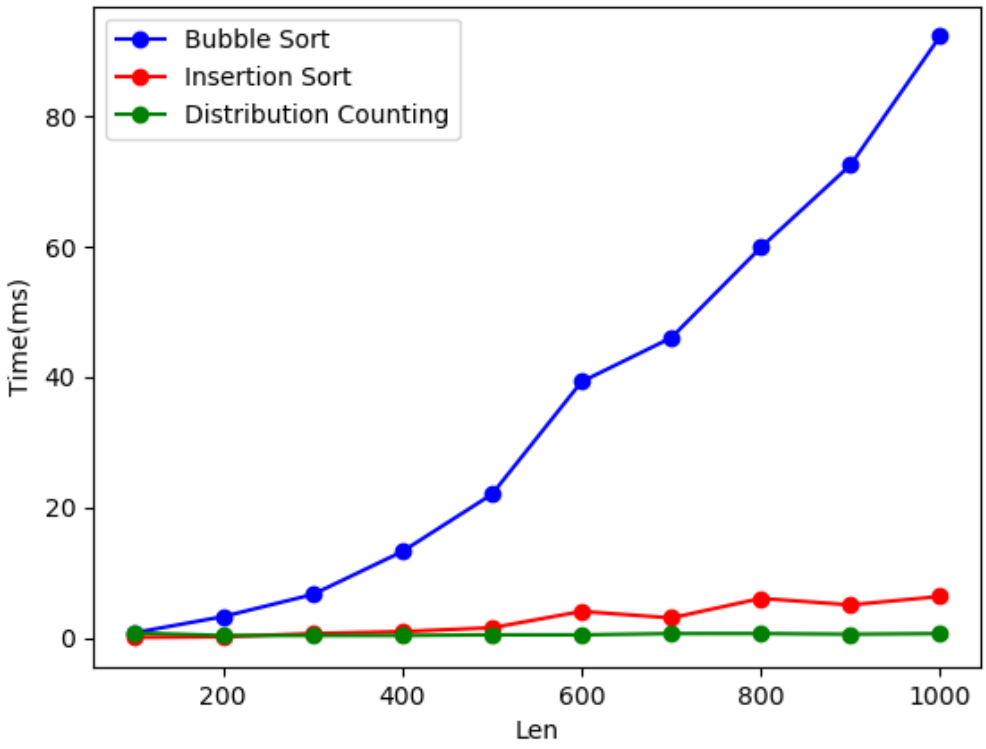
Значение массива от 0 до 100 000

ii. **Среднее время**

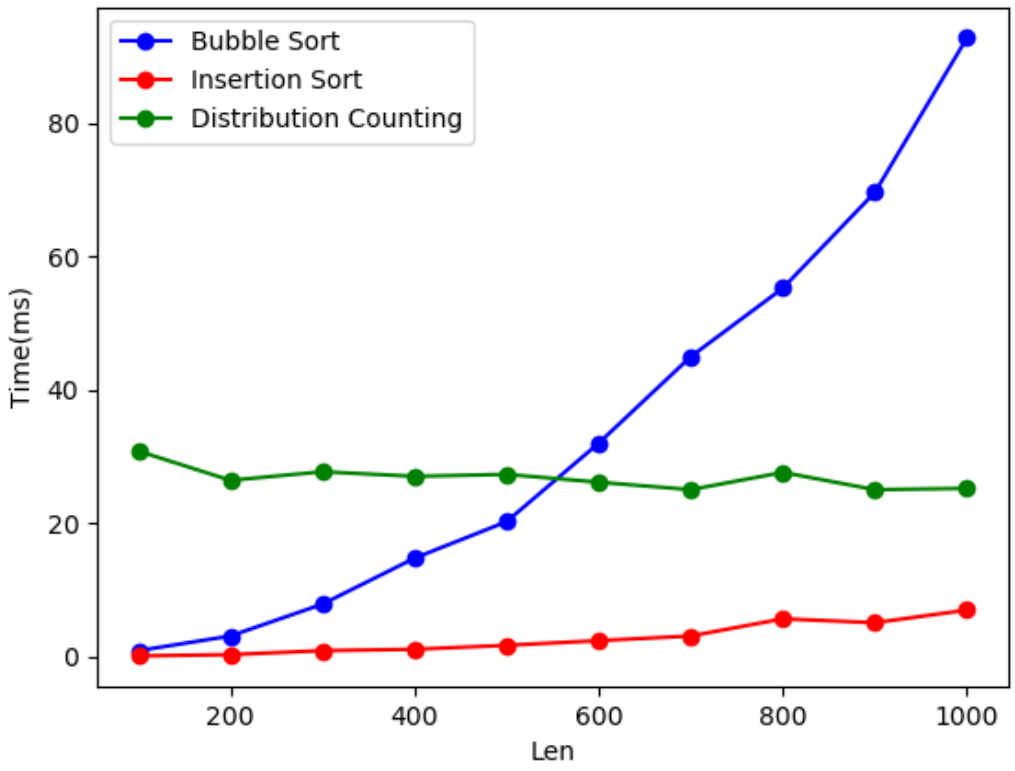
Для сравнения среднего времени работы алгоритмов сортировки массивов были использованы случайно сгенерированные массивы длиной от 100 до 1000 с шагом 100. Значение массива от 0 до 1000 и от 0 до 100 000.

Эксперимент для более точного результата повторялся 10 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице и на рисунке.

	Max_Value = 1000			Max_Value = 100 000		
	Сортировка пузырьком	**Сортировка вставками**	**Сортировка подсчётом**	**Сортировка пузырьком**	**Сортировка вставками**	**Сортировка подсчётом**
100	0.79	0.12	0.79	0.89	0.09	30.81
200	3.29	0.22	0.41	3.09	0.29	26.43
300	6.68	0.69	0.39	7.87	0.88	27.72
400	13.26	0.99	0.40	14.76	1.09	27.03
500	22.05	1.59	0.49	20.24	1.70	27.33
600	39.29	4.08	0.50	32.01	2.37	26.15
700	46.02	3.09	0.71	44.97	3.01	25.04
800	59.88	6.08	0.69	55.25	5.67	27.62
900	72.50	5.09	0.59	69.61	5.13	25.03
1000	92.25	6.38	0.69	92.78	6.98	25.23



Значение массива от 0 до 1000



Значение массива от 0 до 100 000

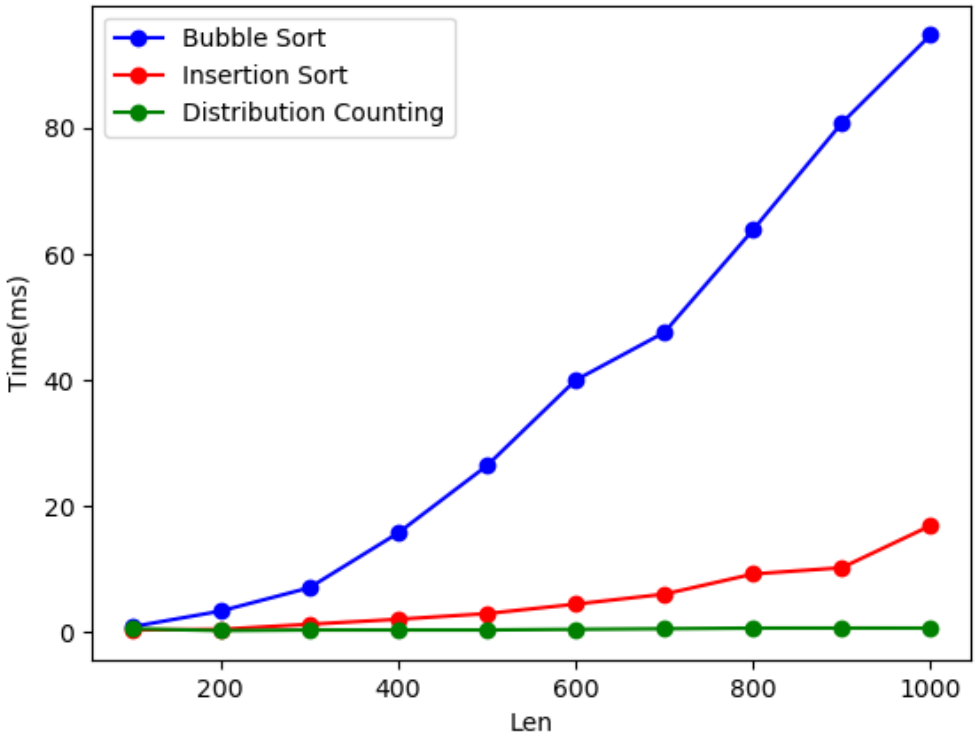
По среднему времени

iii. **Худшее время**

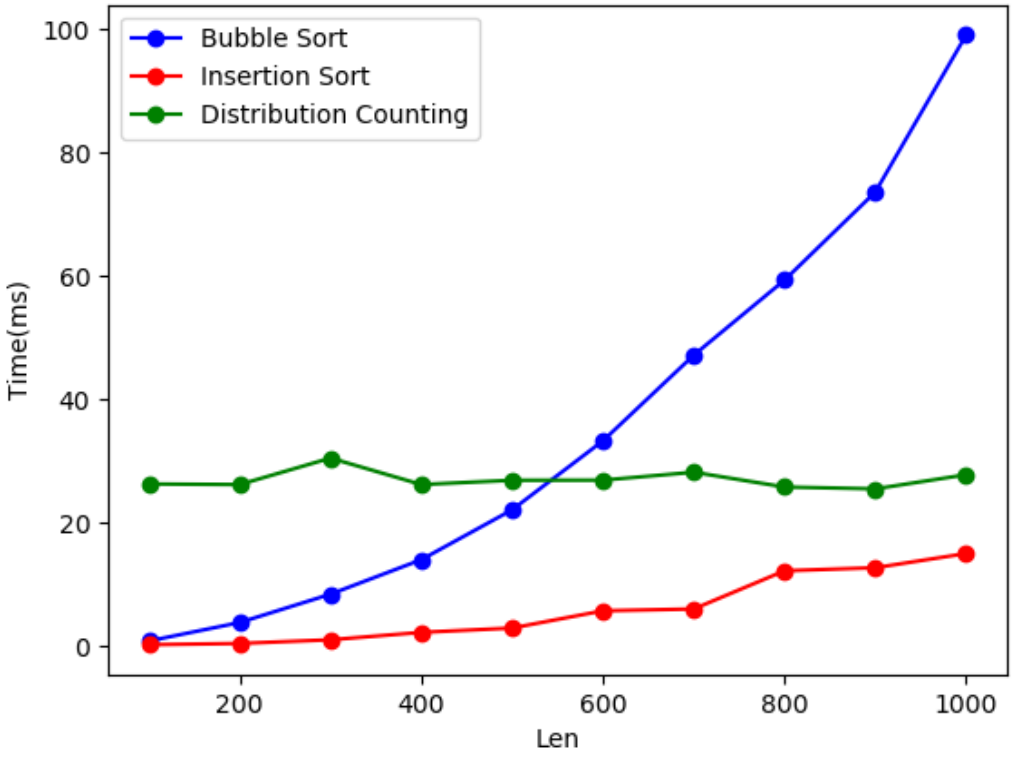
Для сравнения среднего времени работы алгоритмов сортировки массивов были использованы отсортированные в обратном порядке массивы длиной от 100 до 1000 с шагом 100. Значение массива от 0 до 1000 и от 0 до 100 000.

Эксперимент для более точного результата повторялся 10 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице и на рисунке

	Max_Value = 1000			Max_Value = 100 000		
	Сортировка пузырьком	**Сортировка вставками**	**Сортировка подсчётом**	**Сортировка пузырьком**	**Сортировка вставками**	**Сортировка подсчётом**
100	0.90	0.40	0.6	0.89	0.31	26.32
200	3.39	0.49	0.29	3.88	0.49	26.23
300	7.08	1.29	0.40	7.47	1.19	30.51
400	15.76	2.09	0.39	14.06	2.15	26.23
500	26.42	2.99	0.49	24.14	3.01	26.92
600	39.99	4.49	0.59	35.31	5.12	26.94
700	47.57	6.08	0.70	47.17	6.08	26.22
800	63.84	9.27	0.72	59.34	12.27	25.83
900	80.78	10.27	0.70	74.6	12.76	25.53
1000	96.74	16.95	0.69	99.34	15.06	27.82



Значение массива от 0 до 1000



Значение массива от 0 до 100 000

Вывод:

В большинстве случаев алгоритм сортировки вставкой быстрее сортировки пузырьком.

Сортировка подсчётом - зависит от не только количества элементов в массиве но и значения элементов массива. В случае максимального значения, равного длине массива, алгоритм Сортировка подсчётом является наиболее оптимальным из 3-х алгоритмов..

Заключение

- ✓ реализован алгоритмы сортировок массивов: сортировка вставками, пузырьком, подсчётом;
- ✓ оценил трудоёмкости алгоритмов;
- ✓ выполнил сравнительный анализ алгоритмов на основе трудоёмкости;
- ✓ оценил и сравнил эффективности алгоритмов по времени.

В ходе лабораторной работе были изучены и реализованы три алгоритма умножения матриц: классический алгоритм, алгоритм Винограда и его оптимизированный вариант. Сравнительный анализ алгоритмов показал, что алгоритмы Винограда, введя дополнительные векторы, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный и оптимизированный варианты работают быстрее классического алгоритма.