Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет имени Н.Э. Баумана»

Отчет

По лабораторной работе № 3
По курсу «Анализ Алгоритмов»
На тему «Исследование сложности алгоритмов сортировки»

Студент: Козырев Марк
Группа: ИУ7-56
Преподаватель: Волкова Л. Л.

Москва, 2018

Введение

В лабораторной работе изучаются алгоритмы сортировки. Рассмотрены алгоритмы: пузырек, гномья сортировка, выбором.

Цель лабораторной работы - проанализировать трудоемкость алгоритмов и провести сравнительный анализ времени работы алгоритмов для различных массивов

Постановка задачи

1. Ввести модель оценки трудоемкости
2. Реализовать алгоритмы умножения сортировки:
   A. Алгоритм сортировки пузырек
   B. Алгоритм гномьей сортировки
   C. Алгоритм сортировки выбором
3. Провести временные замеры
4. Провести расчет трудоемкости для реализованных алгоритмов
5. Провести сравнительный анализ времени работы алгоритма для разных исходных массивов

1. Аналитическая часть

В данном разделе приведены алгоритмы и составлена модель для вычисления трудоемкости.

1. 1. Описание алгоритмов

В данном разделе приведены описания алгоритмов

1. 1. 1. Алгоритм сортировки пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N - 1 раз. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший - на одну позицию к началу

1. 1. 2. Алгоритм гномьей сортировки

Гномья сортировка похоже на сортировку пузырьком, Поддерживаем указатель на текущий элемент, если он больше предыдущего или он первый — смещаем указатель на позицию вправо, иначе меняем текущий и предыдущий элементы местами и смещаемся влево.

1. 1. 3. Алгоритм сортировки выбором

Шаги алгоритма:

1. Находим номер минимального значения в текущем списке
2. Производим обмен этого значения со значением первой неотсортированный позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. Теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

1. 2. Модель вычислений

Введем следующую модель вычислений:
операции + - ; / < <= > >= = <> [] % имеют стоимость 1.
Логические операции имеют стоимость 1.

1. 2. 1. Оценка трудоемкости цикла for

Инициализация до цикла стоит 2, после выполнения тела цикла, инкрементируется итератор цикла, проверяется условие по стоимости выше. Если в условии содержится выражение, то оно считается как сумма стоимости простых операций выше.

1. 2. 2. Оценка трудоемкости операции if

Переход по условию имеет стоимость 0, проверка условия зависит от выражения самой провеки согласно модели выше.

1. 2. 3. Оценка трудоемкости цикла while

Условие оценивается согласно модели выше.

2. Конструкторская часть

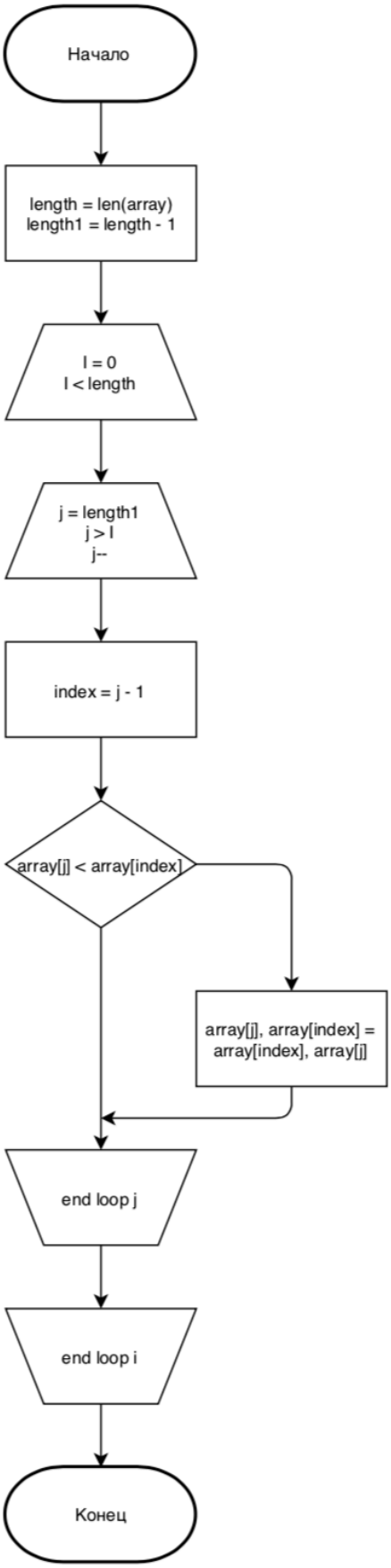

Рис. 1: Схема алгоритма сортировки пузырьком

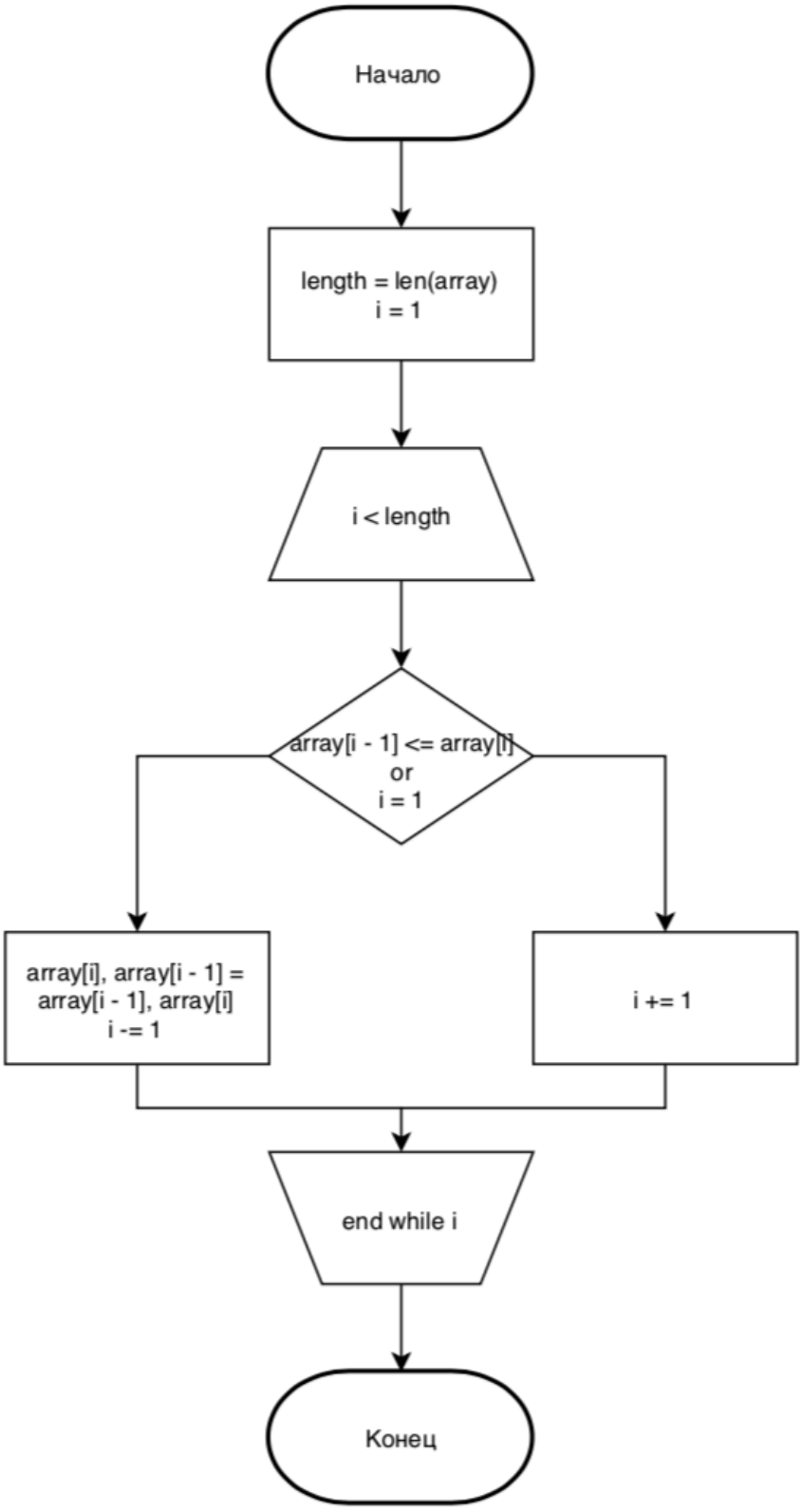

Рис. 2: Схема алгоритма гномьей сортировки

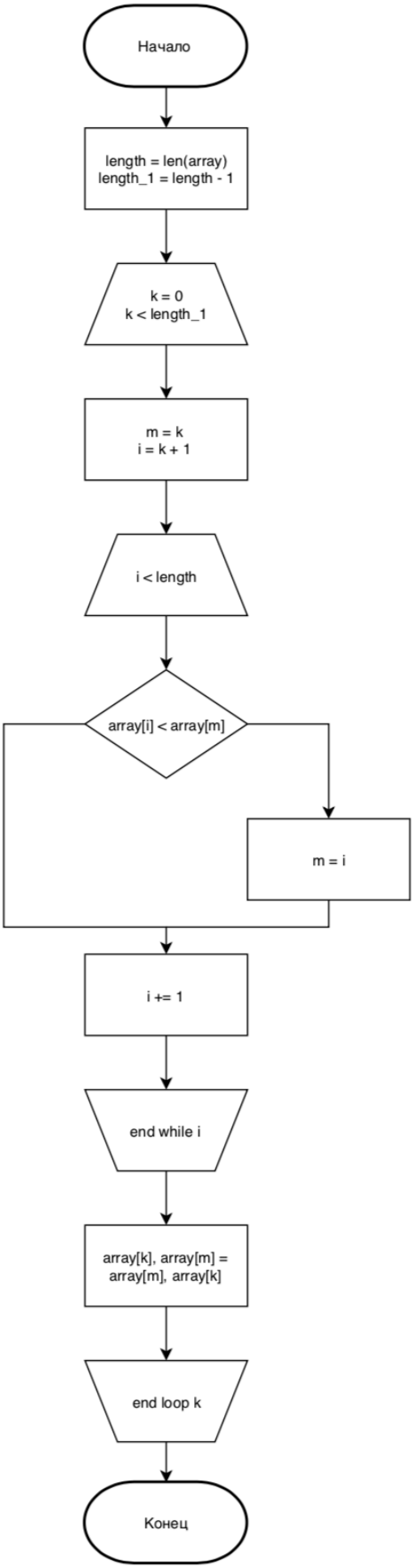

Рис. 3: Схема алгоритма сортировки выбором

3. Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

3. 1. Средства реализации

В данной работе использовался язык Python 3.6. Для измерения времени использовался модуль time, измерения производились в миллисекундах.

3. 2. Листинг кода

```
import random
import tests
from time import time

def main():
  array = [(random.randint(0, 9)) for i in range(100)]
  print(bubble_sort(array))
  print(gnome_sort(array))
  print(vibor_sort(array))
  #tests.test()


def bubble_sort(array):
  length = len(array)
  length1 = length - 1
  start = time()
  #Bad
  #2 + 2n + 12mn
  #Good
  #2 + 2n + 7mn
  for i in range(length): # 2 + n(2 + )
    for j in range(length1, i, -1): # m(2 + )
      index = j - 1 #2
      if array[j] < array[index]: # 3
        array[j], array[index] = array[index], array[j] #5
  end = time() - start
  return end

def gnome_sort(array):
  length = len(array)
  start = time()
  #Good
  #1 + 7n
  #Bad
  #1 + 15n
  i = 1 # 1
  while i < length: # n(1 + )
    if not i or array[i - 1] <= array[i]: #5
      i += 1 # 1
    #Bad
    else:
      array[i], array[i - 1] = array[i - 1], array[i] # 7
      i -= 1 # 1
  end = time() - start
  return end
```

```
46
47 def vibor_sort(array):
48     start = time()
49     length = len(array)
50     length_1 = length - 1
51     #Bad
52     #2 + 10n + 6mn
53     #Good
54     #2 + 10n + 5mn
55     for k in range(length_1): #2 + n(2 + )
56         m = k #1
57         i = k + 1 # 2
58         while i < length: #m(1 + )
59             if array[i] < array[m]: #3
60                 m = i #1
61             i += 1 #1
62         array[k], array[m] = array[m], array[k] #5
63     end = time() - start
64     return end
65
66 if __name__ == '__main__':
67     main()
```

main.py

3. 3. Вычисление трудоемкости алгоритмов

Расчет производился по исходному коду

Оценка трудоемкости пузырька:

Наилучший случай: 7MN + 2N + 2

Наихудший случай: 12MN + 2N + 2

Оценка трудоемкости гномьей сортировки

Наилучший случай: 7N + 1 ~$O(n)$

Наихудший случай: 15N + 1 ~$O(n)$

Оценка трудоемкости выбором

Наилучший случай: 5MN + 10N + 2 ~ $O(n^2)$

Наихудший случай: 6MN + 10N + 2 ~ $O(n^2)$

Согласно справочным материалам [1] трудоемкость алгоритма сортировки выбором:

$$f = O(n^2)$$

4. Экспериментальная часть

В данном разделе будут приведены примеры работы алгоритмов и произведены замеры времени. Замеры времени проводились на компьютере с процессоре 2,4 GHz Intel Core i5 и с оперативной памятью 8 ГБ 1600 MHz DDR3

4. 1. Примеры работы

Имеет массив A с размером 10, A = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] результатам работы алгоритмов будет массив A с размером 10, A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

4. 2. Временные эксперименты

Измерения проводились для квадратных целочисленных матриц с помощью функцией time() из встроенного модуля python time. Замеры времени в секундах. Значения брались из среднего значения результатов 400 экспериментов.

Рандом			
Размер массива	Сортировка пузырьком	Гномья сортировка	Сортировка выбором
1000	0.09127700328826904	0.0001971721649169922	0.07744395732879639
2000	0.38653063774108887	0.0005789995193481445	0.3091164827346802
3000	0.8592923879623413	0.0007803440093994141	0.6957135200500488
4000	1.5480486154556274	0.000928044319152832	1.2563984394073486
5000	2.3974945545196533	0.0010315179824829102	1.959492564201355
6000	3.3943779468536377	0.0012253522872924805	2.7845109701156616
7000	4.498149514198303	0.0014480352401733398	3.611496686935425
8000	5.9052969217300415	0.0016770362854003906	4.706670522689819
9000	7.497711896896362	0.0018863677978515625	6.026497960090637
10000	9.417152285575867	0.002097010612487793	7.811994910240173

Таблица 1: сортировки рандомного массива

Наилучший случай			
Размер массива	Сортировка пузырьком	Гномья сортировка	Сортировка выбором
1000	0.0005899667739868164	1.6927719116210938e-05	0.0006636381149291992
2000	0.0023039579391479492	3.2901763916015625e-05	0.002618074417114258
3000	0.005505084991455078	5.054473876953125e-05	0.006209969520568848
4000	0.0115278959274292	8.249282836914062e-05	0.012591958045959473
5000	0.017376065254211426	9.28640365600586e-05	0.01813650131225586
6000	0.025332093238830566	0.00011229515075683594	0.025850534439086914
7000	0.034795522689819336	0.0001304149627685547	0.03537106513977051
8000	0.04625344276428223	0.0001531839370727539	0.04657852649688721
9000	0.058004140853881836	0.00017142295837402344	0.06086266040802002
10000	0.07182228565216064	0.00019097328186035156	0.07594060897827148

Таблица 2: Сортировки массивов, упорядоченных по возрастанию

Наихудший случай			
Размер массива	Сортировка пузырьком	Гномья сортировка	Сортировка выбором
1000	0.10646653175354004	0.00019359588623046875	0.0755009651184082
2000	0.39382290840148926	0.0003955364227294922	0.2954615354537964
3000	0.8451335430145264	0.0006680488586425781	0.6679928302764893
4000	1.5520271062850952	0.0008581876754760742	1.4983079433441162
5000	2.296653985977173	0.001223921775817871	2.18470299243927
6000	3.1593973636627197	0.0017069578170776367	2.7161351442337036
7000	4.025518417358398	0.0015207529067993164	3.835847496986389
8000	5.186984539031982	0.001788496971130371	4.771177053451538
9000	6.665700435638428	0.001855015754699707	6.145083427429199
10000	8.174327611923218	0.0024938583374023438	7.650256514549255

Таблица 3: Сортировки массивов, упорядоченный по убыванию

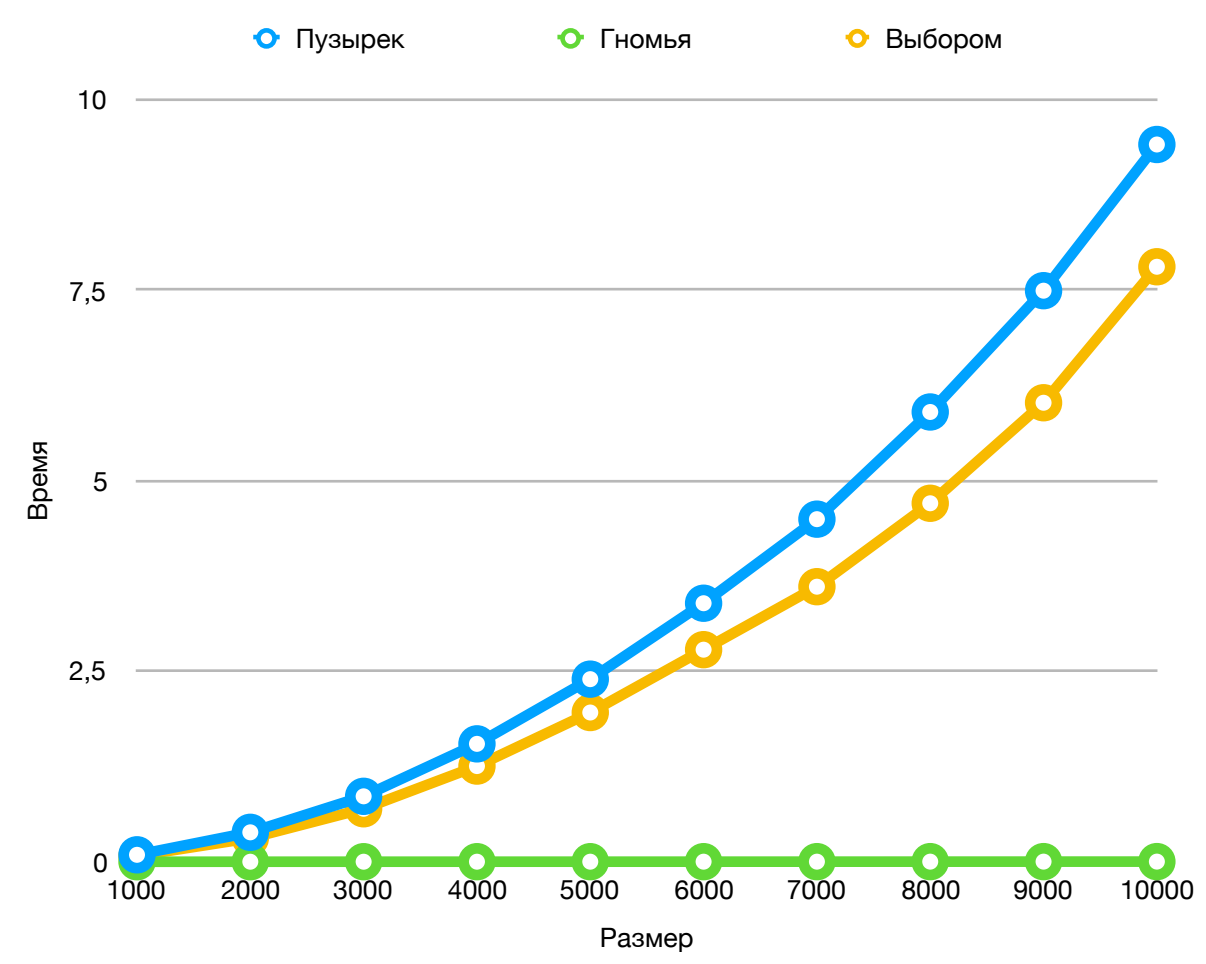

Рис. 4: График зависимости времени от размера массива (рандомный массив)

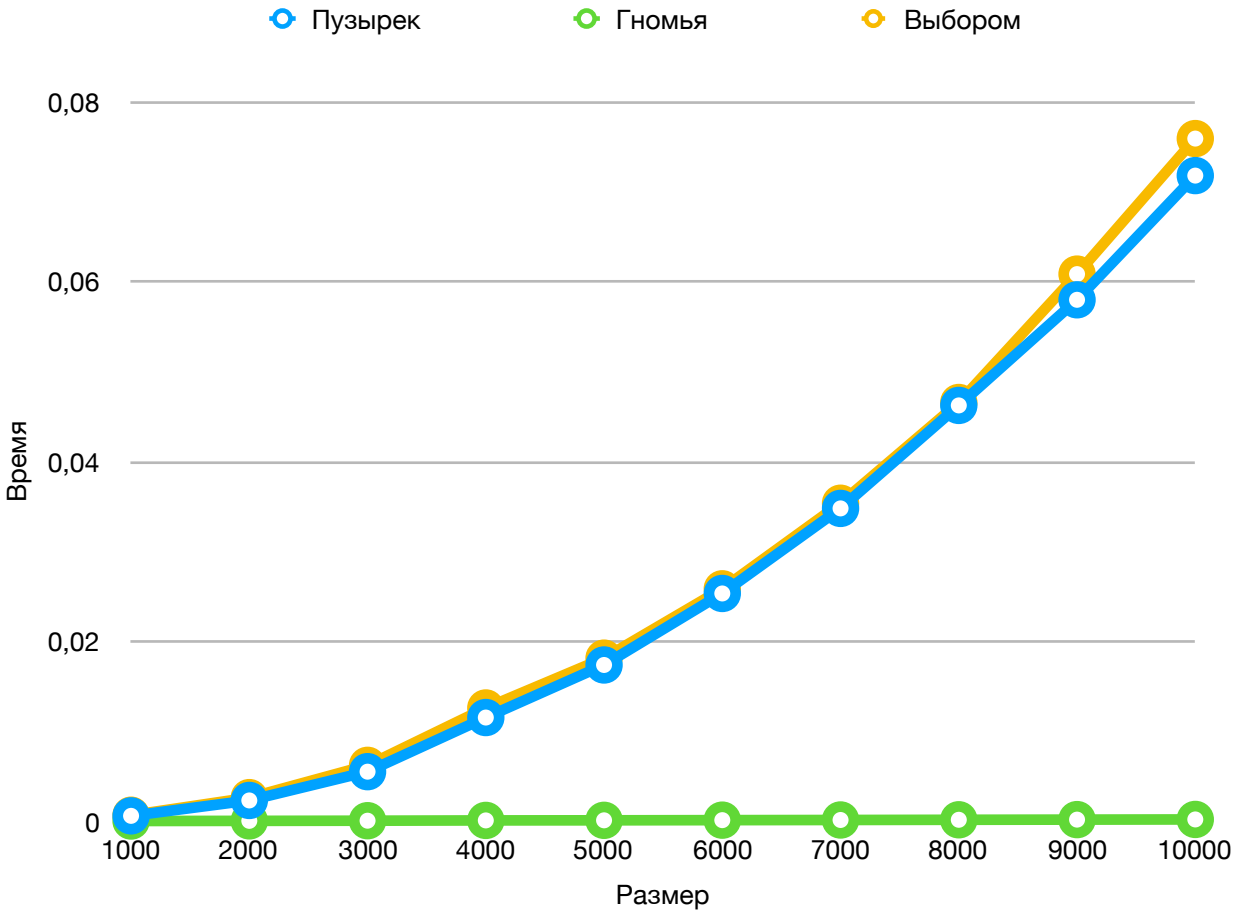

Рис. 5: График зависимости времени от размера массива (отсортированный массив)

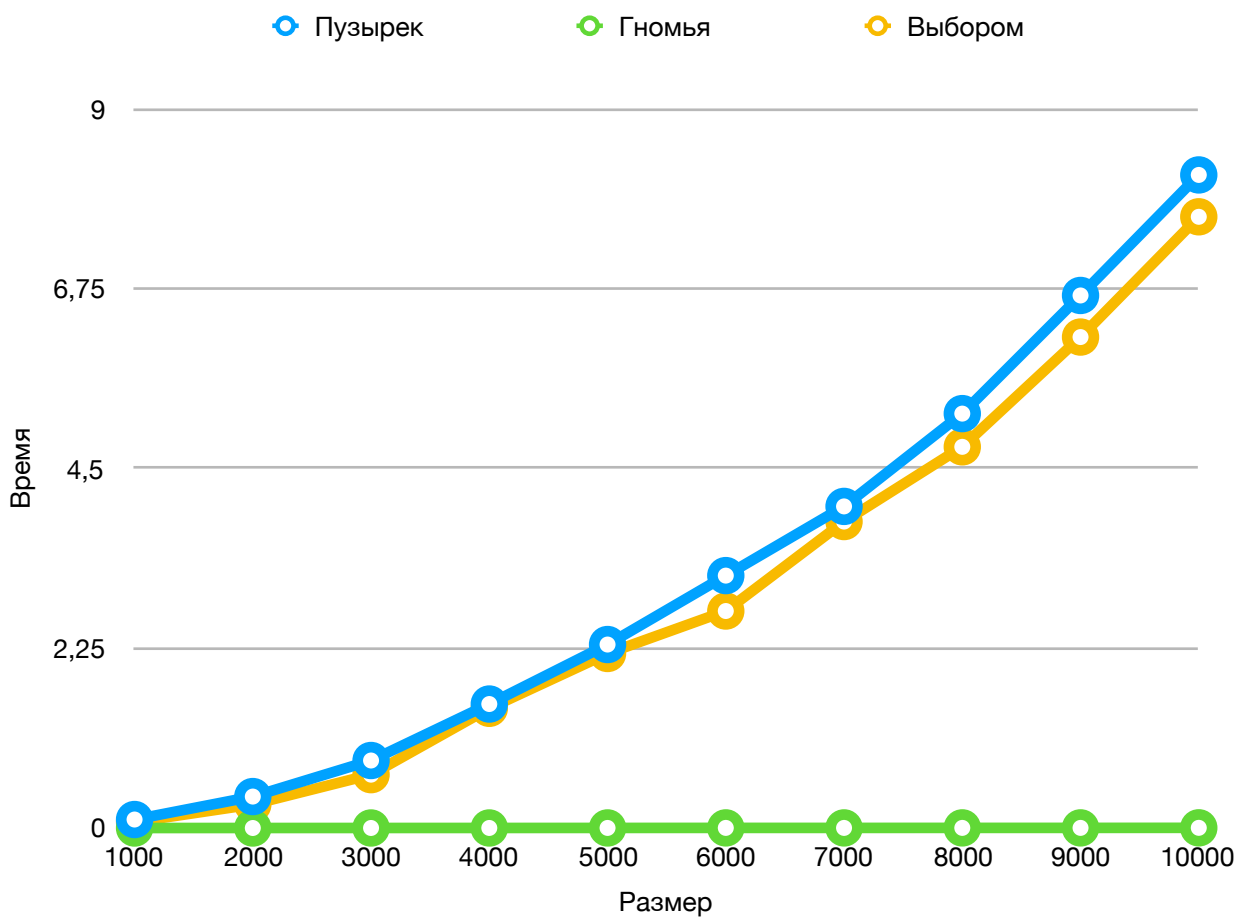

Рис. 6: График зависимости времени от размера массива (убывающий массив)

В результате проведенных испытаний алгоритмов было установлено, что:

1. Алгоритм сортировки пузырьком самый затратный алгоритм сортировки, что совпадает с высчитанной трудоемкостью

2. Гномья сортировка является быстродействующей сортировкой в сравнении с пузырьком или сортировкой выбором

Заключение

В ходе лабораторной работы были реализованы три алгоритма сортировки: пузырек, гномья, выбором, проведены вычисления трудоемкости и выполнен сравнительный анализ сортировок. Были получены навыки оптимизации кода на python.

Источники

1. Левитин А. В. Алгоритмы: введение в разработку и анализ, 2006, Издательский дом Вильямс