Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа №3

Исследование сложности сортировок

Студент группы ИУ7-55Б,
Руднев К. К.,

Преподаватель,
Волкова Л. Л.,
Строганов Ю. В.

2019 г.

Оглавление

1 Аналитическая часть **4**
1.1 Описание алгоритмов 4
1.1.1 Коктейльная сортировка 4
1.1.2 Сортировка выбором 4
1.1.3 Гномья сортировка 4
1.2 Вывод 4

2 Конструкторская часть **5**
2.1 Разработка алгоритмов 5
2.2 Вывод 7

3 Технологическая часть **8**
3.1 Средства реализации 8
3.2 Требования к программному обеспечению 8
3.3 Листинг кода 8
3.4 Вывод 9

4 Экспериментальная часть **10**
4.1 Примеры работы 10
4.2 Сравнительный анализ на основе экспериментальных данных 11
4.3 Оценка трудоёмкости 13
4.3.1 Модель оценки трудоёмкости 13
4.3.2 Гномья сортировка 14
4.3.3 Сортировка выбором 14
4.3.4 Коктейльная сортировка 14
4.4 Вывод 14

Введение

На текущий момент существует огромное количество разнообразных сортировок. Эти алгоритмы необходимо уметь сравнивать, чтобы выбирать наиболее походящий в конкретном случае.
Эти алгоритмы сравниваются по:

- времени сортировки;
- затратам памяти.

Цель работы: изучение применений алгоритмов сортировки, обучение расчету трудоёмкости алгоритмов.

Глава 1

Аналитическая часть

В рамках раздела будет дано описание гномьей, коктейльной и сортировки выбором.

1.1 Описание алгоритмов

Сортировка массива - одна из самых популярных операций, проводимых над массивом. Алгоритмы реализуют упоядочивание элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки.

1.1.1 Коктейльная сортировка

На каждом шаге основного цикла рассматривается массив array[Left]-array[Right], после выполнения двух внутренних циклов минимальный и максимальный элемент в исходном массиве перетекают к краям, минимальный в — array[Left], максимальный — в array[Right] [1].

1.1.2 Сортировка выбором

На каждом i-ом шаге алгоритма находим i-ый минимальный элемент и меняем его местами с i-ым элементом в массиве [2]. Таким образом будет получен массив, отсортированный по неубыванию.

1.1.3 Гномья сортировка

Алгоритм находит первое место, где два соседних элемента стоят в неправильном порядке и меняет их местами [3]. Он пользуется тем фактом, что обмен может породить новую пару, стоящую в неправильном порядке, только до или после переставленных элементов. Он не допускает, что элементы после текущей позиции отсортированы, таким образом, нужно только проверить позицию до переставленных элементов.

1.2 Вывод

В этом разделе были рассмотрены описания гномьей, коктейльной и сортировки выбором.

Глава 2

Конструкторская часть

В рамках раздела будут даны схемы алгоритмов для гномьей (рисунок 2.1), коктейльной (рисунок 2.2) и сортировки выбором (рисунок 2.3).

2.1 Разработка алгоритмов

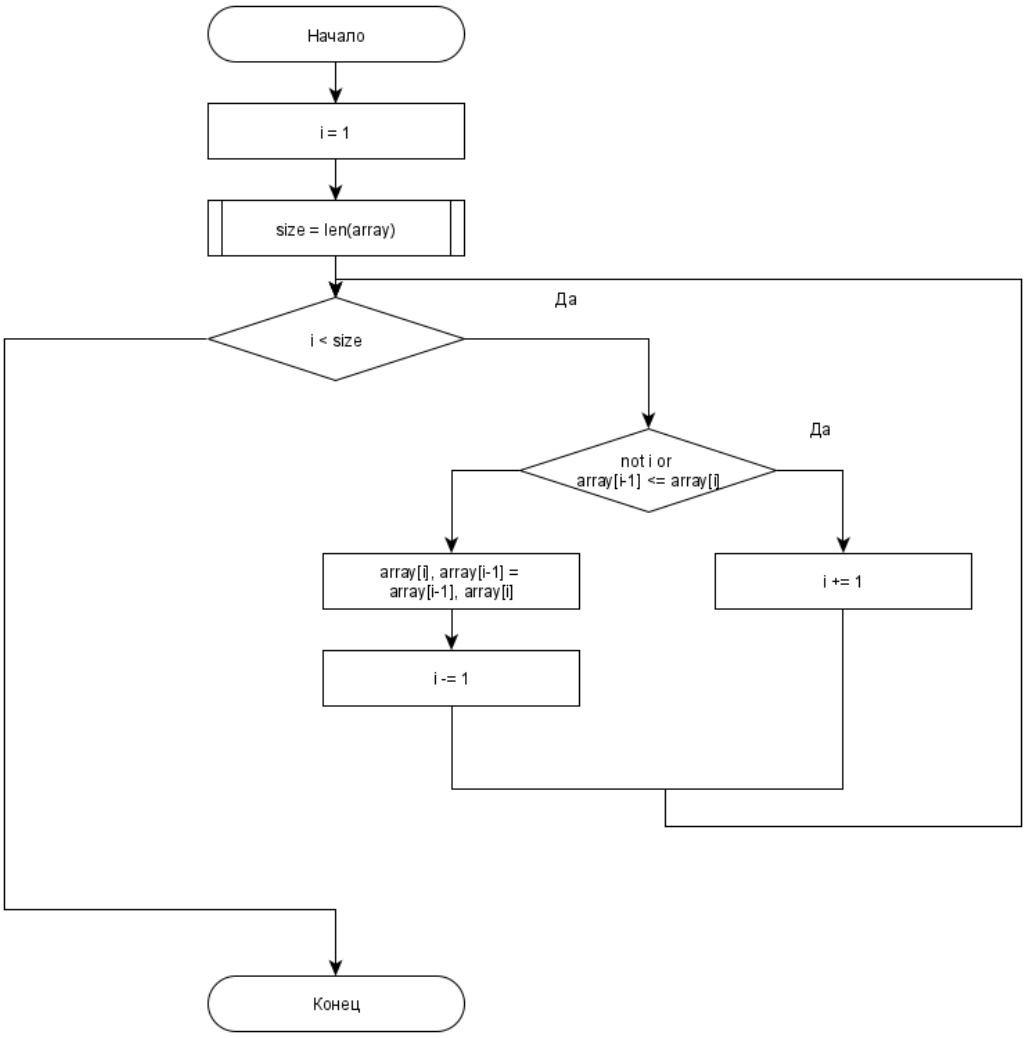

Рис. 2.1: Гномья сортировка

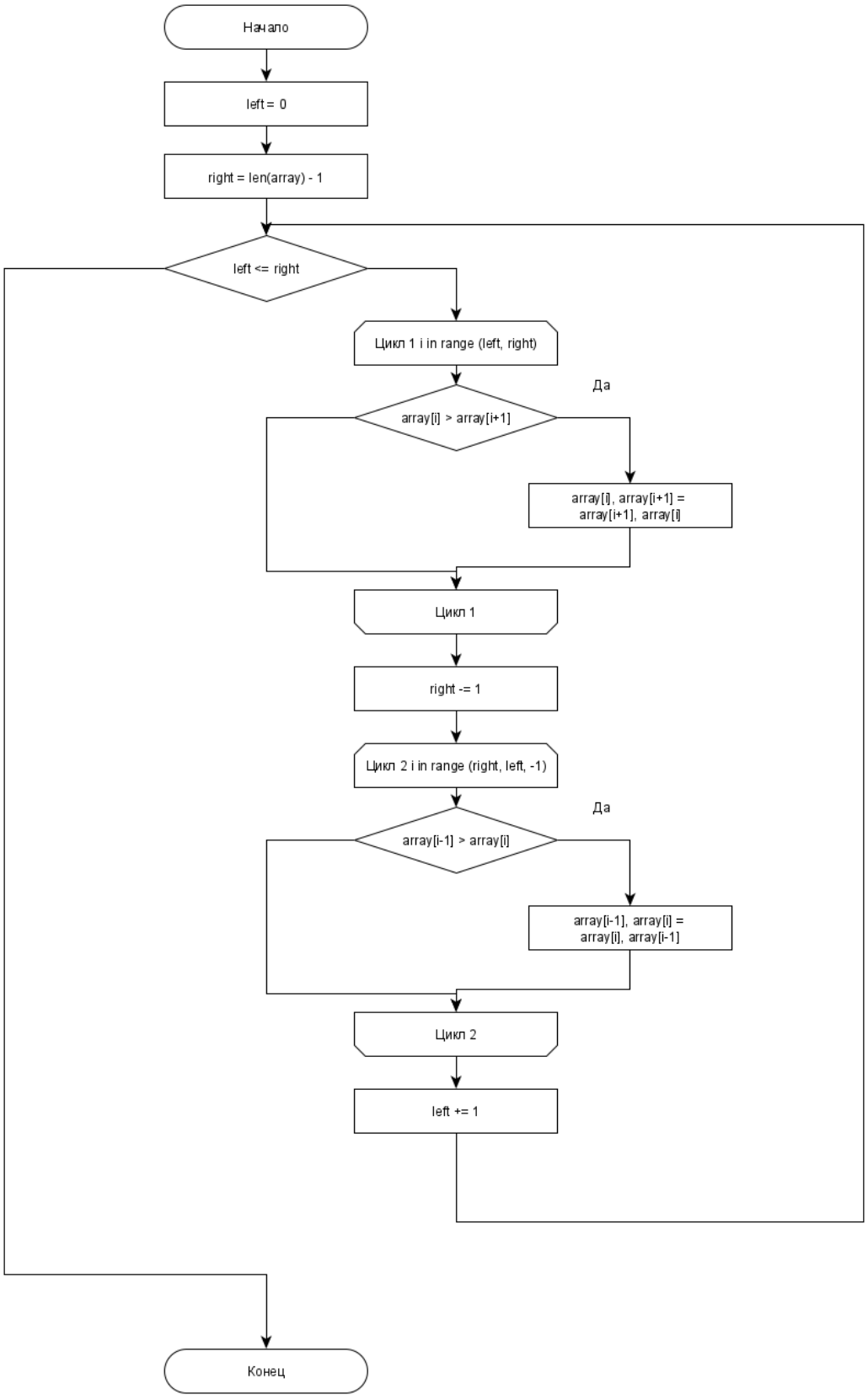

Рис. 2.2: Коктейльная сортировка

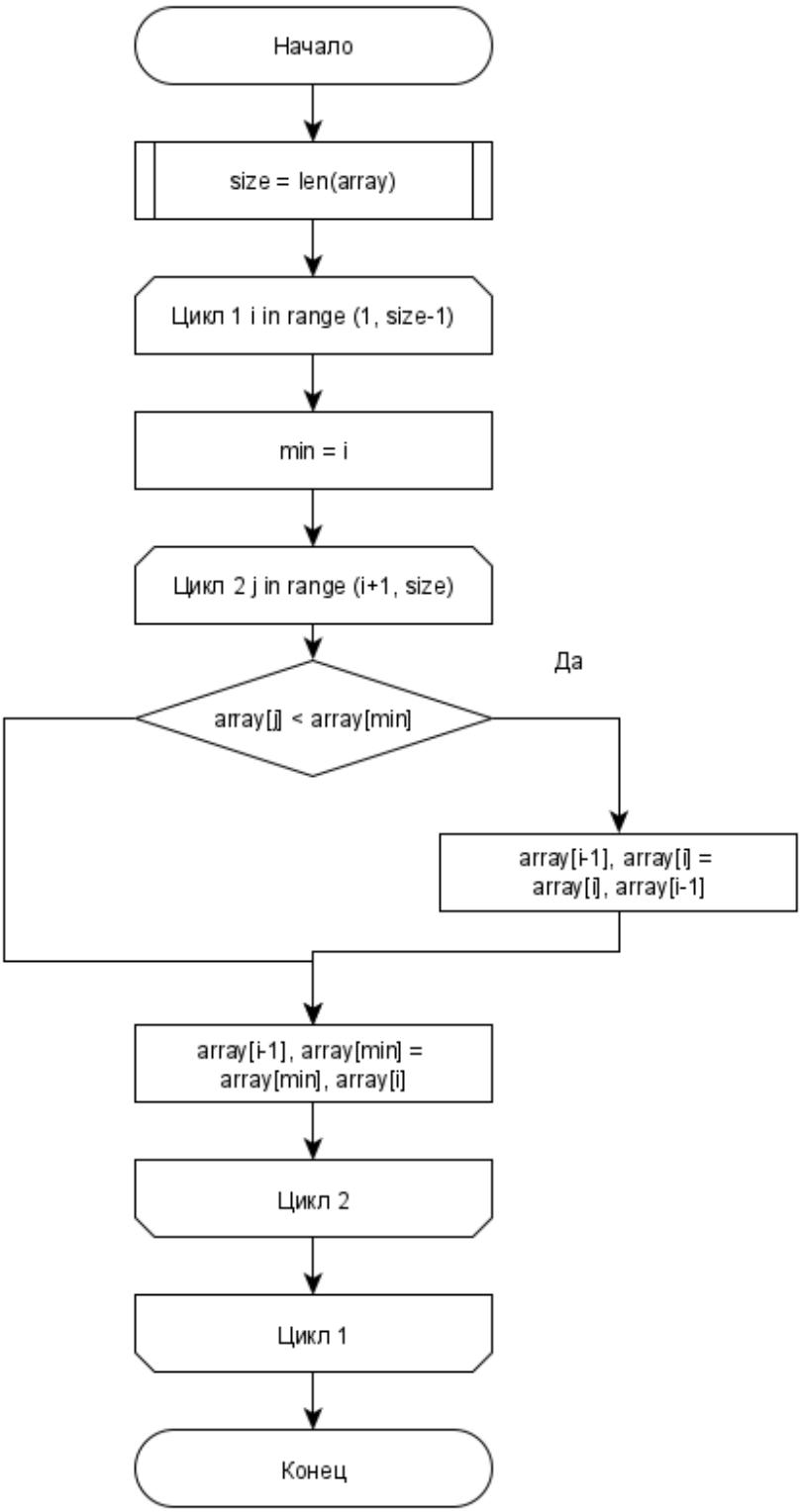

Рис. 2.3: Сортировка выбором

2.2 Вывод

В этом разделе были рассмотрены схемы алгоритмов коктейльной, гномьей и сортировки выбором.

Глава 3

Технологическая часть

В рамках раздела будут описаны инструментарии разработки, выбор среды, требования к ПО. Также будут представлены листинги конкретных реализаций алгоритмов сортировок на рисунках 3.1-3.3.

Замеры времени были произведены на: Intel(R) Core(TM) i7-8565U, 4 ядра, 8 логических процессоров.

3.1 Средства реализации

Для реализации алгоритмов использовался язык программирования Python 3.8.0 и среда разработки PyCharm Community Edition 2019.3.1 by JetBrains. У меня есть определенный опыт работы с данным языком, которого будет достаточно для реализации текущей лабораторной работы, а среда разработки имеет бесплатную комьюнити версию и удобный интерфейс, упрощающий разработку приложения/скрипта.

Замер времени реализован с помощью функции process_time() библиотеки time. Измеряется время исполнения кода чистого алгоритма (без учета времени на генерацию данных и т.п.).

3.2 Требования к программному обеспечению

На вход программа должна получать массив данных, который сортируется тремя алгоритмами (гномья сортировка, коктейльная сортировка, а также сортировка выбором).

На выход программа должна выдавать отсортированный по неубыванию массив данных всеми тремя алгоритмами.

3.3 Листинг кода

На листинге 3.1 представлена реализация коктейльной сортировки. На листинге 3.2 представлена реализация гномьей сортировки. На листинге 3.3 представлена реализация сортировки выбором.

Листинг 3.1: Коктейльная сортировка

```
def cocktail(array):
    left = 0
    right = len(array)-1

    while left <= right:
        for i in range(left, right):
```

```
                if array[i] > array[i+1]:
                    array[i], array[i+1] = array[i+1], array[i]
            right = right - 1

            for i in range(right, left, -1):
                if array[i-1] > array[i]:
                    array[i-1], array[i] = array[i], array[i-1]
            left = left + 1

        return array
```

Листинг 3.2: Гномья сортировка

```
def gnome(array):
    i = 1
    size = len(array)

    while i < size:
        if not i or array[i - 1] <= array[i]:
            i += 1
        else:
            array[i], array[i - 1] = array[i - 1], array[i]
            i -= 1

    return array
```

Листинг 3.3: Сортировка выбором

```
def choice(array):
    size = len(array)

    for i in range (size - 1):
        min = i

        for j in range (i + 1, size):
            if array[j] < array[min]:
                min = j

        array[i], array[min] = array[min], array[i]

    return array
```

3.4 Вывод

В рамках раздела были предъявлены требования к программному обеспечению. На основании их были разработаны и представлены конкретные реализации всех трёх алгоритмов сортировок.

Глава 4

Экспериментальная часть

В рамках раздела будут проведены тесты работы программы, представленные на рисунках 4.1-4.3. Также будут проведены эксперименты, результаты которых представлены на рисунках 4.4-4.6

4.1 Примеры работы

```
original array : [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
gnome sort:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
choise sort:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
coctail sort:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Рис. 4.1: Тест работы сортировок на сортированных массивах

```
original array : [390, 351, 312, 273, 234, 195, 156, 117, 78, 39]
gnome sort:  [39, 78, 117, 156, 195, 234, 273, 312, 351, 390]
choise sort:  [39, 78, 117, 156, 195, 234, 273, 312, 351, 390]
coctail sort:  [39, 78, 117, 156, 195, 234, 273, 312, 351, 390]
```

Рис. 4.2: Тест работы сортировок на обратно сортированных массивах

```
original array : [877, -426, -507, 808, 564, 955, -120, 534, 481, -794]
gnome sort:  [-794, -507, -426, -120, 481, 534, 564, 808, 877, 955]
choise sort:  [-794, -507, -426, -120, 481, 534, 564, 808, 877, 955]
coctail sort:  [-794, -507, -426, -120, 481, 534, 564, 808, 877, 955]
```

Рис. 4.3: Тест работы сортировок на случайных массивах

4.2 Сравнительный анализ на основе экспериментальных данных

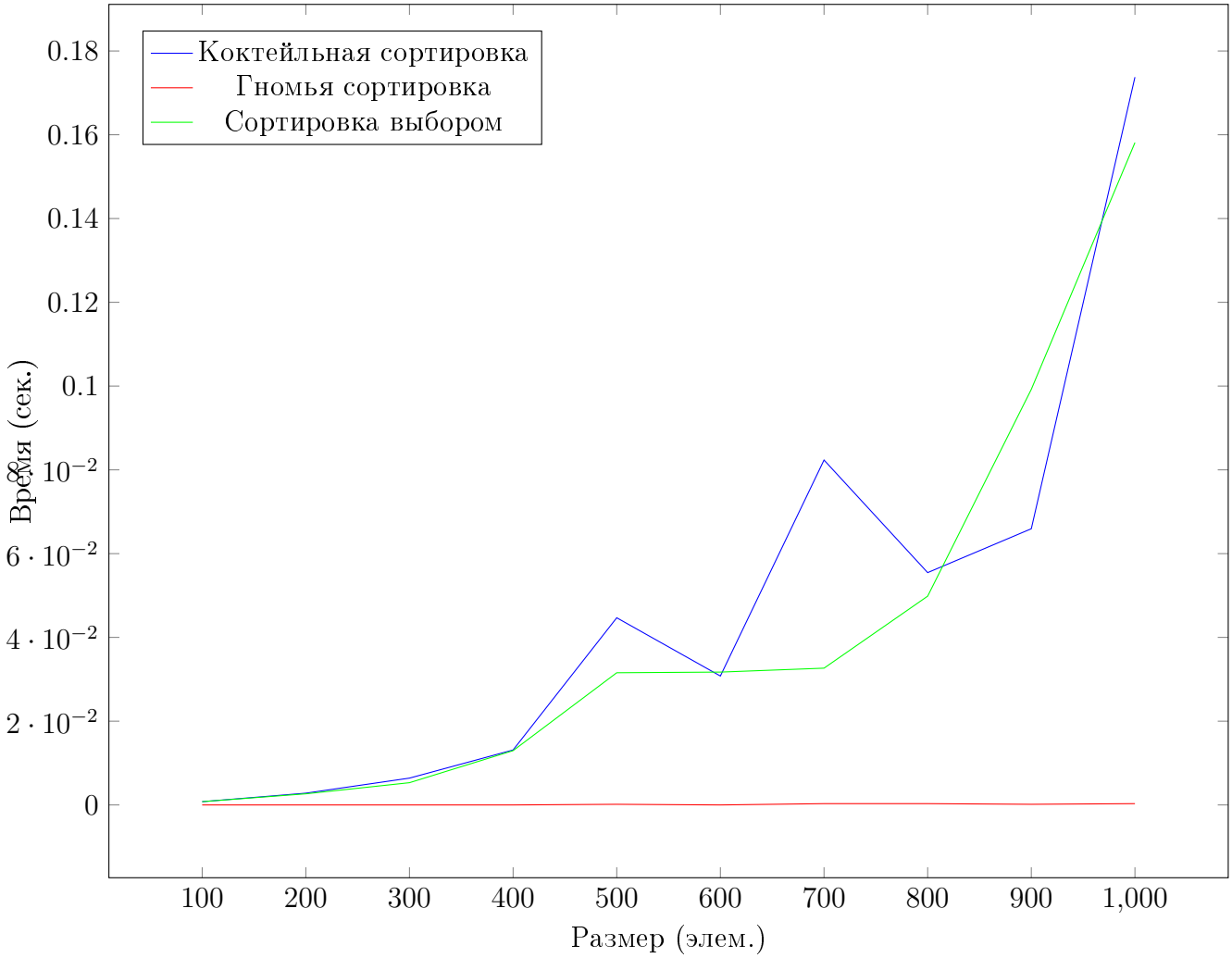

Рис. 4.4: Время выполнения сортировки на упорядоченном массиве

Для всех выбранных сортировок случай отсортированного по возрастанию массива (при условии сортировки по возрастанию) является лучшим случаем. В данной ситуации быстрейшей является гномья сортировка с разницей до 900%.

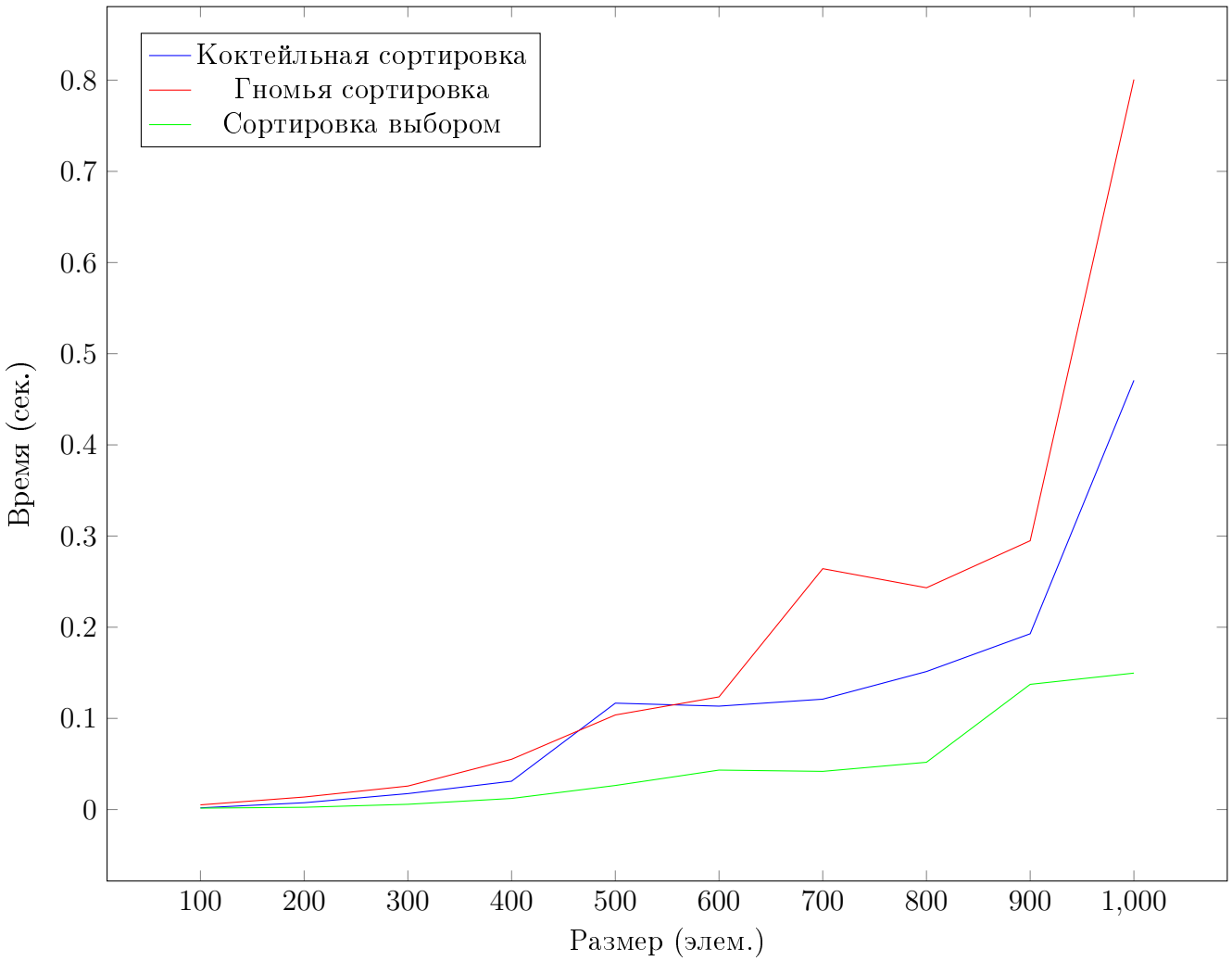

Рис. 4.5: Время выполнения сортировки на обратно упорядоченном массиве

Для всех выбранных сортировок случай отсортированного по убыванию массива (при условии сортировки по возрастанию) является худшим случаем. В данной ситуации быстрейшей является сортировка выбором с разницей до 780%.

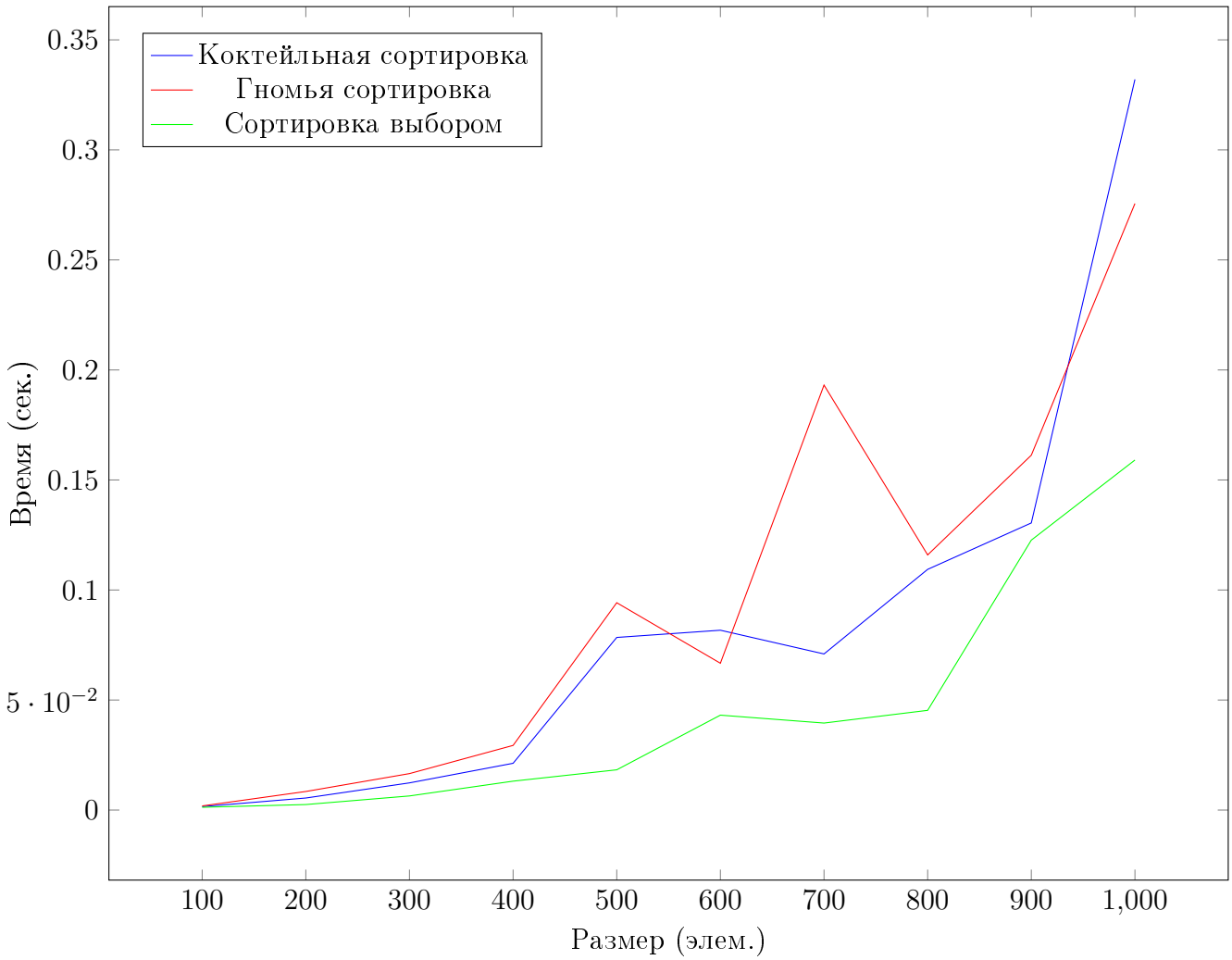

Рис. 4.6: Время выполнения сортировке на случайном массиве

В случае случайного массива быстрейшей сортировкой является сортировка выбором с разницей до 233%.

4.3 Оценка трудоёмкости

Считаем, что сортируются массивы размером N.

4.3.1 Модель оценки трудоёмкости

Введём систему оценки трудоемкости.

1. Объявление переменной/массива без определения имеет трудоёмкость 0

2. Операторы +,-,*,/,=, а также +=,-=,// имеют трудоёмкость 1

3. Оператор доступа по индексу [] имеет трудоёмкость 1

4. Логические операции имеют трудоёмкость 1

5. Цикл имеет трудоёмкость 2+N*(2+T), где N - кол-во итераций, T - трудоёмкость тела цикла

4.3.2 Гномья сортировка

Лучший случай: отсортированный массив - получаем это: 2+N-1+N*(1+1+1+1+1)+1 = 7*N+2 $\sim$ O(N)

Худший случай: отсортированный по убыванию массив - получаем это: 2+(N^2-1)*7+(N^2-1)*9 $\sim$ O(N^2)

4.3.3 Сортировка выбором

Лучший случай: отсортированный массив - получаем это: сортировка выбором - устойчивая сортировка, значит: N-1 повторений внешнего цикла, N/2 повторений внутреннего цикла. Тогда $\sim$ O(N^2)

Худший случай: отсортированный по убыванию массив - получаем это: N-1 повторений внешнего цикла, N/2 повторений внутреннего цикла. Тогда (N-1)*(4+N/2*(4)+5) = (N-1)*(4+2*N+5) = 9*N-9+2*N^2-2*N = 2*N^2+7*N-9 $\sim$ O(N^2)

4.3.4 Коктейльная сортировка

Лучший случай: отсортированный массив - получаем это: учитывая, что массив отсортирован, имеем один проход по массиву с линейной сложностью $\sim$ O(N) [1]

Худший случай: отсортированный по убыванию массив - получаем это: учитывая, что массив отсортирован по убыванию, имеем сложность $\sim$ O(N^2) [1]

4.4 Вывод

Все сортировки в качестве худшего случая показали квадратичную сложность. В лучшем случае самой быстрой с линейной сложностью оказалась гномья сортировка.

В ходе эксперимента выяснилось, что гномья сортировка позволяет добиться выигрыша до 900% при сортировке уже отсортированного массива.

Сортировка выбором быстрее до 780% на обратно отсортированном массиве, а также быстрее на случайном массиве с выигрышем до 233%.

Заключение

В ходе лабораторной работы были изучены алгоритмы сортировки массива: гномья, выбором и коктейльная. Выполнено сравнение всех рассматриваемых алгоритмов. Приведены их трудоемкости.

В ходе экспериментов выяснилось, что из рассмотренных сортировок гномья сортировка самая быстрая из предложенных в лучшем случае и выигрывает во времени до 900% на тестовых данных размерностью до 1000 элементов в массиве.

Быстрейшей сортировкой в худшем случае и среднем случае оказалась сортировка выбором. Дает выигрыш до 780% в худшем и до 233% в среднем на тестовых данных размерностью до 1000 элементов в массиве.

Литература

[1] Н. Вирт Алгоритмы и структуры данных. М., Издат-во "Вильямс 1998г.

[2] Д. Кнут. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. М., "Мир 1978 г., переиздание - М.,Изд-во "Вильямс 2000 г.

[3] А. Ахо, Дж. Э. Хопкрофт, Д. Ульман Структуры данных и алгоритмы. М., Изд-во "Вильямс 2000 г.