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 сортирвки»

Щербатюк Дарья, ИУ7-54

Москва, 2017

Оглавление

Постановка задачи

Реализовать алгоритмы сортировки:

1. Гномья сортировка
2. Сортировка простым выбором
3. Сортировка Шелла

Рассчитать сложность алгоритмов и провести временные эксперименты

Блок-схемы

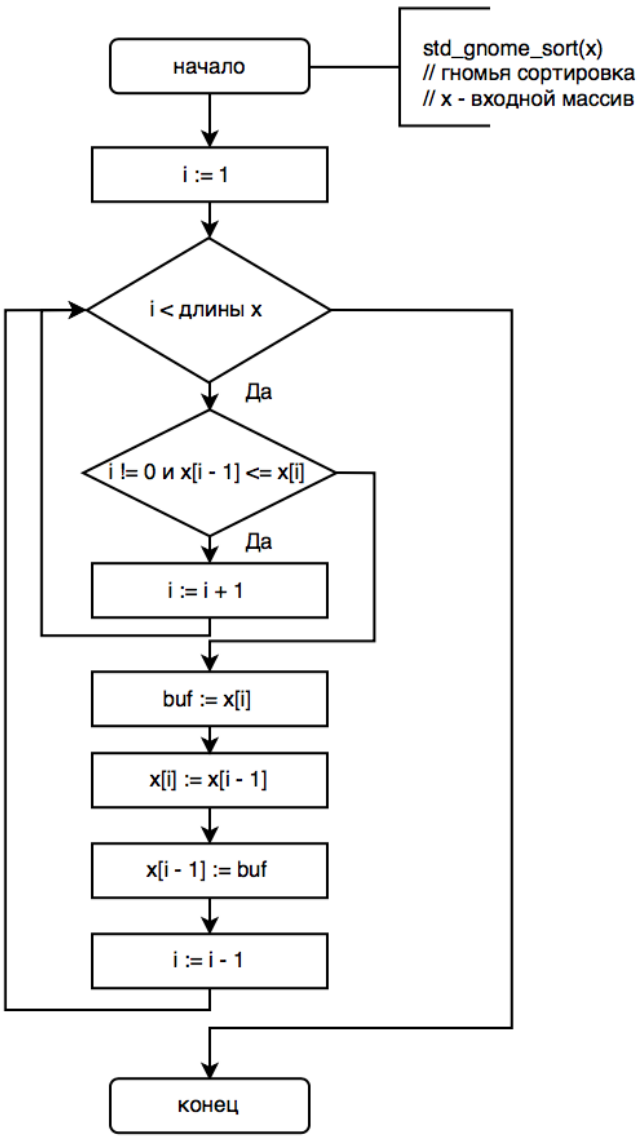

Рис. 1: Блок-схема гномьей сортировки

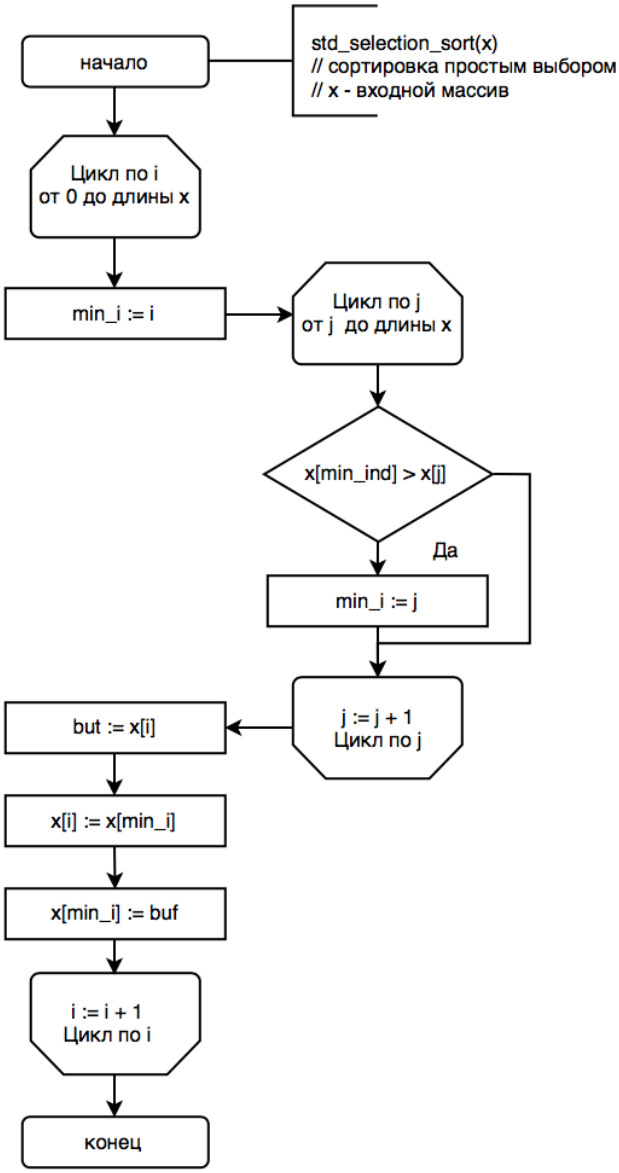

Рис. 2: Блок-схема сортировки простым выбором

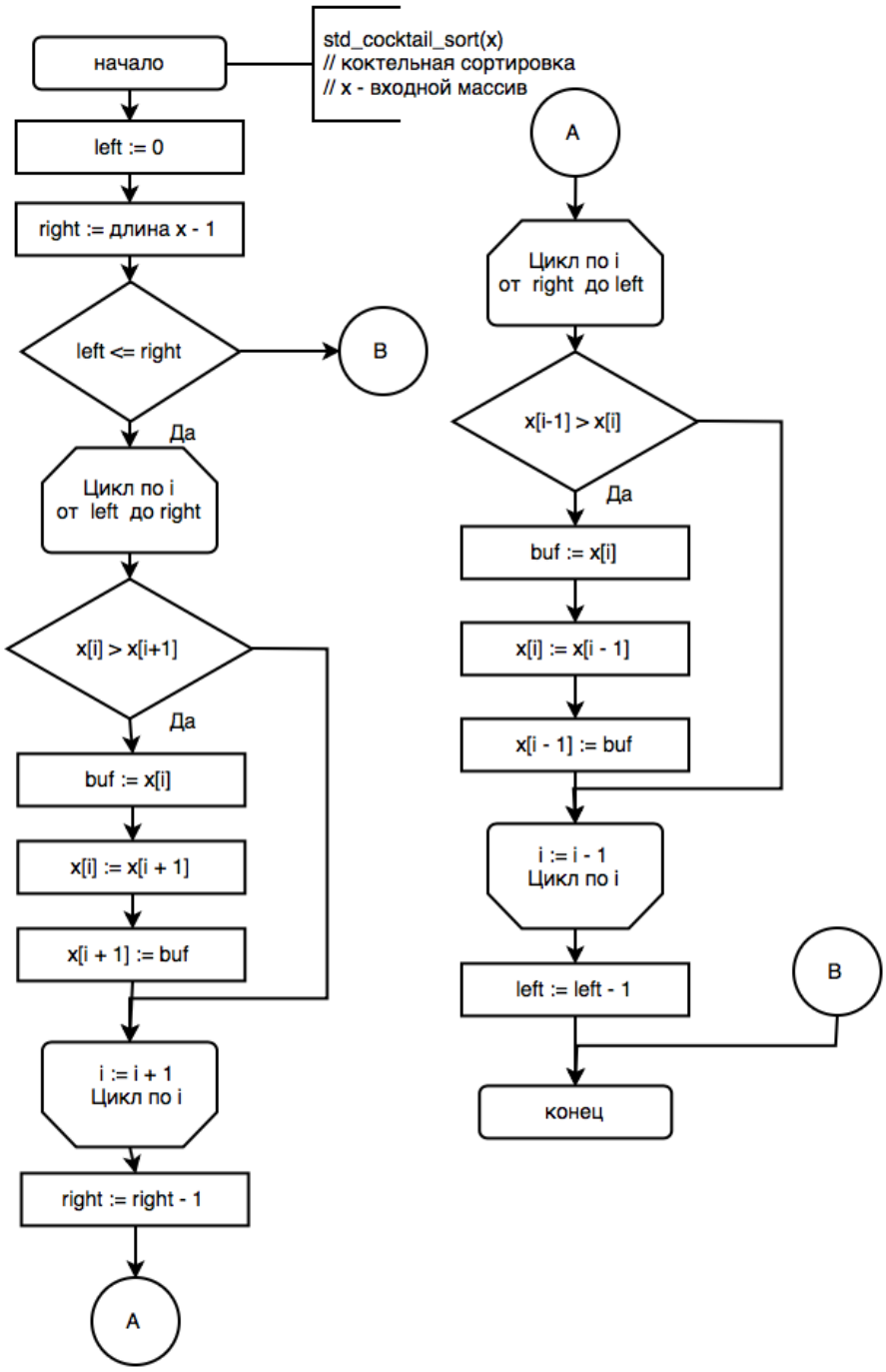

Рис. 3: Блок-схема коктельной сортировки

Модель вычислений

Введём модель вычисления, используемую при оценках трудоёмкости:

1. вызов метода объекта класса имеет трудоёмкость 1;
2. объявление переменной/массива/структуры без определения имеет трудоёмкость 0;
3. операторы +, -, *, /, =, а также ++ и – имеют трудоёмкость 1;
4. условный оператор (без условий внутри) имеет трудоёмкость 0;
5. логические операции имеют трудоёмкость 1;

6. оператор цикла имеет трудоёмкость 1 + n(3 + T), где n – это число повторений цикла, T – трудоёмкость тела цикла;

7. одно присваивание до цикла (1 операция), внутри цикла присваивание, сравнение и инкремент (3 операции).

8. вызов функции имеет трудоёмкость 0, так как функции, объявленные внутри класса, компилятор рассматривает как inline и подставляет сразу вместо вызова код.

9. Fn(n) – часть трудоёмкости, зависящая только от размера входа;

10. P – часть трудоёмкости, зависящая от конкретного входа, значений переменных.

Рассчет сложности алгоритмов

Гномья сортировка:

1. Лучший случай(сортированный массив) :
   $n + n(1 + 1 + 1 + 1 + 1) + 1 = 6 * n + 1 \sim O(n)$

2. Худший случай(сортированный по убыванию) : $n + 13n + 1 = 14 * n + 1 \sim O(n)$

Сортировка простым выбором:

1. Лучший случай(сортированный массив) : $O(n^2)$

2. Худший случай(сортированный по убыванию) : $O(n^2)$

Коктейльная сортировка:

1. Лучший случай(сортированный массив) : $O(n)$

2. Худший случай(сортированный по убыванию) : $O(n^2)$

Листинг

gnome_sort.py:

```
def std_gnome_sort(x):
        i = 1
        while i < len(x):
                if not i or x[i - 1] <= x[i]:
                        i += 1
                else:
                        x[i], x[i - 1] = x[i - 1], x[i]
                        i -= 1

        return x

def opt_gnome_sort(x):
        i, j, size = 1, 2, len(x)
        while i < size:
                if x[i - 1] <= x[i]:
                        i, j = j, j + 1
                else:
                        x[i - 1], x[i] = x[i], x[i - 1]
                        i -= 1
                        if i == 0:
                                i, j = j, j + 1

        return x
```

selection_sort.py:

```
def std_selection_sort(x):
        for i in range(len(x)):
                min_ind = i
                for j in range(i, len(x)):
                        if x[min_ind] > x[j]:
                                min_ind = j
                x[i], x[min_ind] = x[min_ind], x[i]

        return x

def opt_selection_sort(x):
        for i, e in enumerate(x):
                mn = min(range(i, len(x)), key=x.__getitem__
                    )
                x[i], x[mn] = x[mn], e
        return x
```

winograd_multi_matrix:

```
def std_cocktail_sort (x):
        left = 0
        right = len(x)-1

        while left <= right:
```

```
6                       for i in range(left, right):
7                               if x[i] > x[i+1]:
8                                       x[i], x[i+1] = x[i+1], x[i]
9                       right = right - 1
10
11                      for i in range(right, left, -1):
12                              if x[i-1] > x[i]:
13                                      x[i-1], x[i] = x[i], x[i-1]
14                      left = left + 1
15
16              return x
17
18      def opt_cocktail_sort(x):
19              for i in range(len(x)//2):
20                      swap = False
21                      for j in range(1+i, len(x)-i):
22                              if x[j] < x[j-1]:
23                                      x[j], x[j - 1] = x[j - 1], x
                                            [j]
24                                      swap = True
25                      if not swap:
26                              break
27                      swap = False
28                      for j in range(len(x)-i-1, i, -1):
29                              if x[j] < x[j-1]:
30                                      x[j], x[j - 1] = x[j - 1], x
                                            [j]
31                                      swap = True
32                      if not swap:
33                              break
34
35              return x
```

Временные эксперименты

Измерения производились для целых массивов

Размер массива	Сортированный	Сортированный в обратном порядке	Случайный
100	0.024 | 0.015	0.337 | 0.015	0.169 | 0.015
200	0.046 | 0.029	1.302 | 0.029	0.676 | 0.029
300	0.113 | 0.079	3.137 | 0.073	1.988 | 0.067
400	0.106 | 0.063	5.609 | 0.062	3.003 | 0.063
500	0.151 | 0.081	19.762 | 0.082	5.204 | 0.082
600	0.199 | 0.128	13.871 | 0.115	8.015 | 0.126
700	0.234 | 0.142	19.339 | 0.142	10.280 | 0.144
800	0.219 | 0.133	24.594 | 0.133	12.752 | 0.135
900	0.249 | 0.151	30.226 | 0.150	15.669 | 0.153
1000	0.270 | 0.169	38.046 | 0.168	19.415 | 0.172

Замеры времени в мс (среднее из 10 замеров) для Гномьей сортировки | Оптимизированной Гномьей сортировки

Размер массива	Отсортированный	Отсортированный в обратном порядке	Случайный
100	0.451 | 0.489	0.450 | 0.500	0.454 | 0.464
200	1.665 | 1.594	1.690 | 1.658	1.697 | 1.519
300	4.006 | 3.367	3.878 | 3.415	3.877 | 3.376
400	7.468 | 6.557	7.356 | 6.573	7.542 | 6.488
500	11.619 | 10.077	11.554 | 10.010	11.292 | 9.844
600	16.615 | 14.419	17.026 | 14.119	16.475 | 13.697
700	23.426 | 19.965	23.826 | 19.883	22.193 | 19.238
800	29.130 | 25.033	29.828 | 26.019	28.881 | 25.383
900	38.086 | 32.439	38.023 | 33.397	38.096 | 33.032
1000	53.348 | 41.303	54.440 | 41.439	48.975 | 41.408

Замеры времени в мс (среднее из 10 замеров) для Сортировки простым выбором | Оптимизированной Сортировки простым выбором

Размер массива	Отсортированный	Отсортированный в обратном порядке	Случайный
100	0.600 | 0.012	0.649 | 0.012	0.658 | 0.012
200	1.951 | 0.020	2.292 | 0.020	2.165 | 0.020
300	4.784 | 0.039	5.793 | 0.035	5.516 | 0.040
400	8.952 | 0.050	10.139 | 0.050	9.418 | 0.051
500	14.913 | 0.069	16.899 | 0.069	15.825 | 0.067
600	21.402 | 0.070	25.281 | 0.068	23.544 | 0.075
700	31.062 | 0.083	35.196 | 0.080	32.391 | 0.081
800	39.372 | 0.097	46.493 | 0.093	42.907 | 0.094
900	50.386 | 0.118	59.354 | 0.114	54.929 | 0.123
1000	67.438 | 0.146	81.112 | 0.139	75.065 | 0.170

Замеры времени в мс (среднее из 10 замеров) для Коктейльной сортировки | Оптимизированной коктейльной сортировки

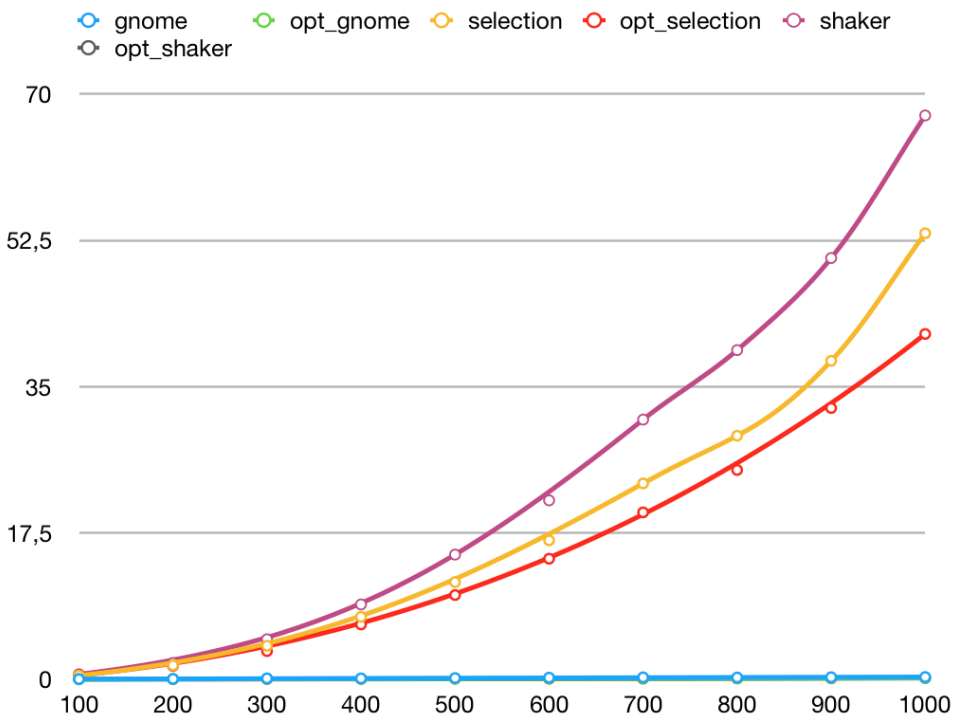

Рис. 4: Графики временных замеров для сортированного массива

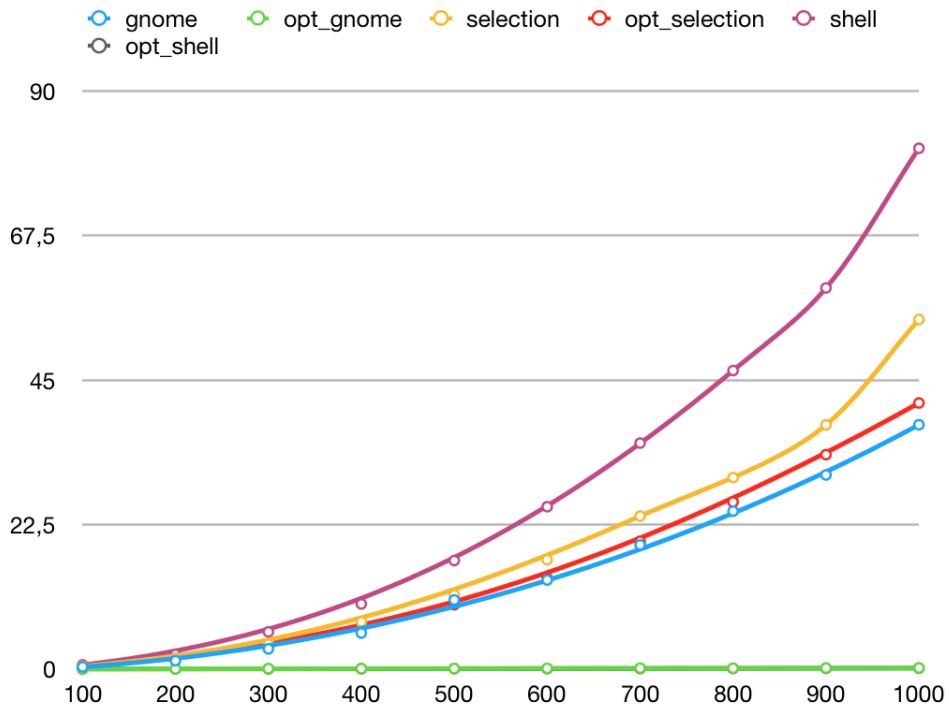

Рис. 5: Графики временных замеров для отсортированного в обратном порядке массива

Выводы

В результате проведенных испытаний алгоритмов было установлено, что:

1. Гномья сортировка является одной из самых быстрых на отсортированном и отсортированным обратно массиве. Она может сравниться только с

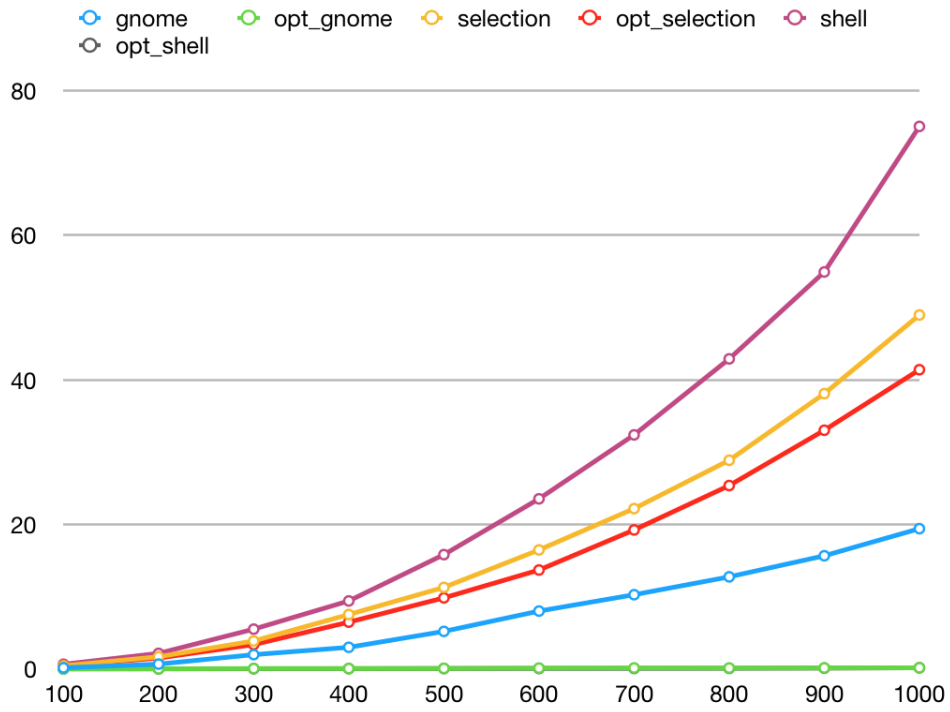

Рис. 6: Графики временных замеров для случайного массива

оптимированной коктейльной сортировкой, использующей преимущества Python.

2. На больших размерах массивов неоптимизированная коктейльная сортировка начинает сильно проигрывать даже неоптимизированной сортировке простым выбором. В таблице представлены результаты для целочисленного элемента длиной в 10 000 элементов. Обе сортировки не оптимизированы.

	Отсортированный	Отсортированный в обратном порядке	Случайный
Selection	4.715 sec	4.851 sec	5.500 sec
Cocktail	6.453 sec	18.148 sec	12.086 sec

3. Коктейльная и гномья сортировки являются обменными сортировками, и, как в следствие, улучшениями пузырьковой сортировки.

4. Сортировка простым выбором может иметь неустойчивую реализацию, то есть может менять относительный порядок сортируемых элементов, имеющих одинаковые ключи(значения). Приведенные выше реализации являются устойчивыми.

5. Время работы сортировки сильно зависит от ее реализации на данном языке.

Заключение

В ходе лабораторной работы были изучены 3 сортровки: гномья, простым выбором и коктейьная. Они были улучшены с использованием средств языка Python.