

Санкт-Петербургский Государственный Технический Университет  
(Технологический институт)

Кафедра системного анализа и информационных технологий

### **Лабораторная работа №4**

Выполнили:

Леякин А. А., Люцай В.Н.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

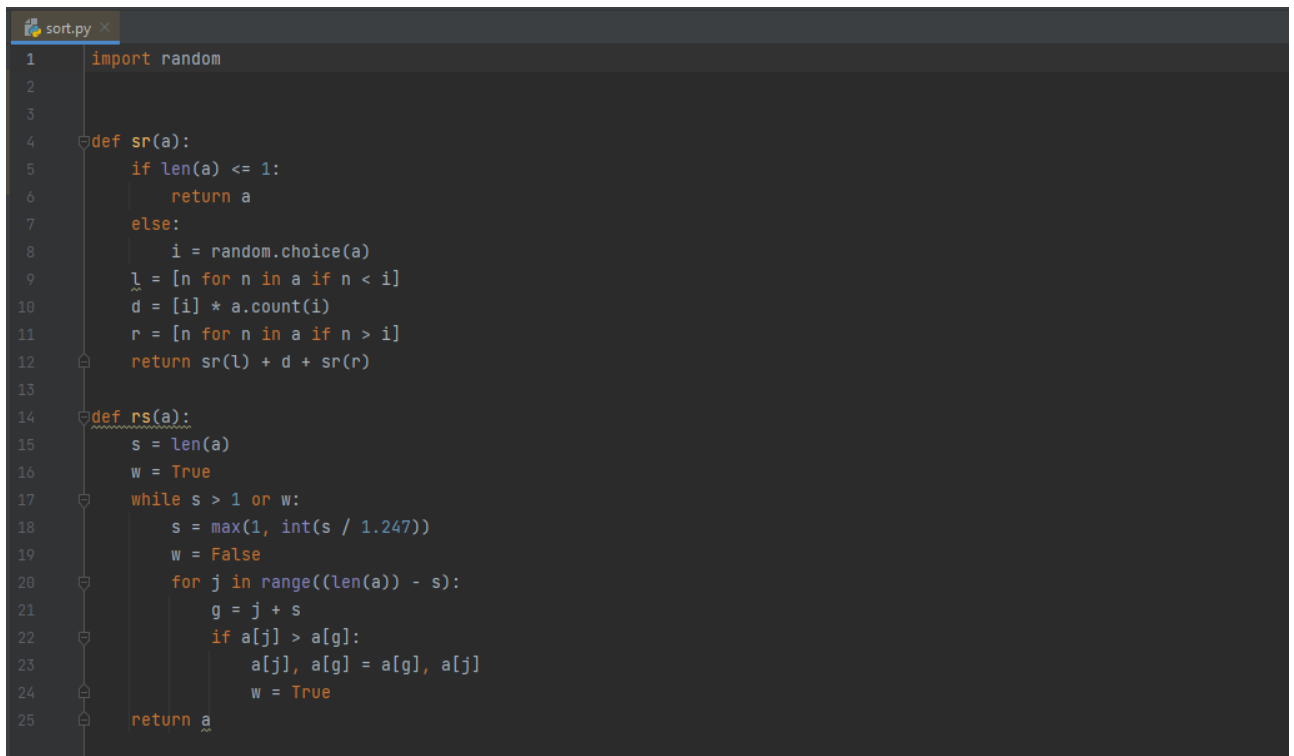
## Ход работы

### Задание 1

“Написать программу с функциями для быстрой сортировки и сортировки расчёской. Использовать данные функции и программу как модуль в другой программе. Пользователь выбирает один из двух методов сортировки. Оценить время выполнения программы с помощью модуля `timeit`.”

### Выполнение:

Мы написали программу с функциями для быстрой сортировки и сортировки расчёской. Код программы представлен ниже (рис.1) и добавлен в репозиторий (наименование файла: “sort.py”).



```
1  import random
2
3
4  def sr(a):
5      if len(a) <= 1:
6          return a
7      else:
8          i = random.choice(a)
9          l = [n for n in a if n < i]
10         d = [i] * a.count(i)
11         r = [n for n in a if n > i]
12         return sr(l) + d + sr(r)
13
14  def rs(a):
15      s = len(a)
16      w = True
17      while s > 1 or w:
18          s = max(1, int(s / 1.247))
19          w = False
20          for j in range((len(a)) - s):
21              g = j + s
22              if a[j] > a[g]:
23                  a[j], a[g] = a[g], a[j]
24                  w = True
25      return a
```

рис.1

Данная программа содержит две функции: `sr` - быстрая сортировка, `rs` - сортировка расчёской.

Далее мы написали программу, которая использует данные функции как модуль. В этой программе пользователь выбирает один из двух предложенных методов сортировки и задает массив для дальнейшей сортировки выбранным методом. Программа, соответствуя введенным данным, производит сортировку заданного массива одним из двух методов. Также она производит оценку времени, затраченного на выполнение программы, с помощью модуля timeit. Код программы представлен ниже (рис.2) и добавлен в репозиторий (наименование файла: “time.py”).

```
time.py
1 import sort
2 import timeit
3
4
5 print('Доступные методы сортировки:'); print(' 1. Быстрая      2. Расчёска'); print()
6 q = input('Введите название метода сортировки, который хотите использовать: ')
7 while q not in ('Быстрая', 'быстрая', 'Расчёска', 'расчёска'):
8     q = input('Данный метод не поддерживается программой, введите поддерживаемый метод сортировки: ')
9 print()
10
11 x=int(input('Введите количество чисел в массиве: '))
12 a=[0]*x
13 print()
14 for i in range(x):
15     print('Введите число', i+1, 'элемента массива:')
16     a[i]=int(input())
17 print(); print('Задан массив:', a); print()
18
19 set_up='''sort.sr(a)'''
20 set_up1='''sort.rs(a)'''
21
22 if q in ('Быстрая', 'быстрая'):
23     print('Результат быстрой сортировки:'); print(sort.sr(a)); print()
24     print('Время затраченное на выполнение программы:', timeit.timeit(setup=set_up, number=100000, globals=globals()))
25
26 if q in ('Расчёска', 'расчёска'):
27     print('Результат сортировки методом "Расчёска":'); print(sort.rs(a)); print()
28     print('Время затраченное на выполнение программы:', timeit.timeit(setup=set_up1, number=100000, globals=globals()))
```

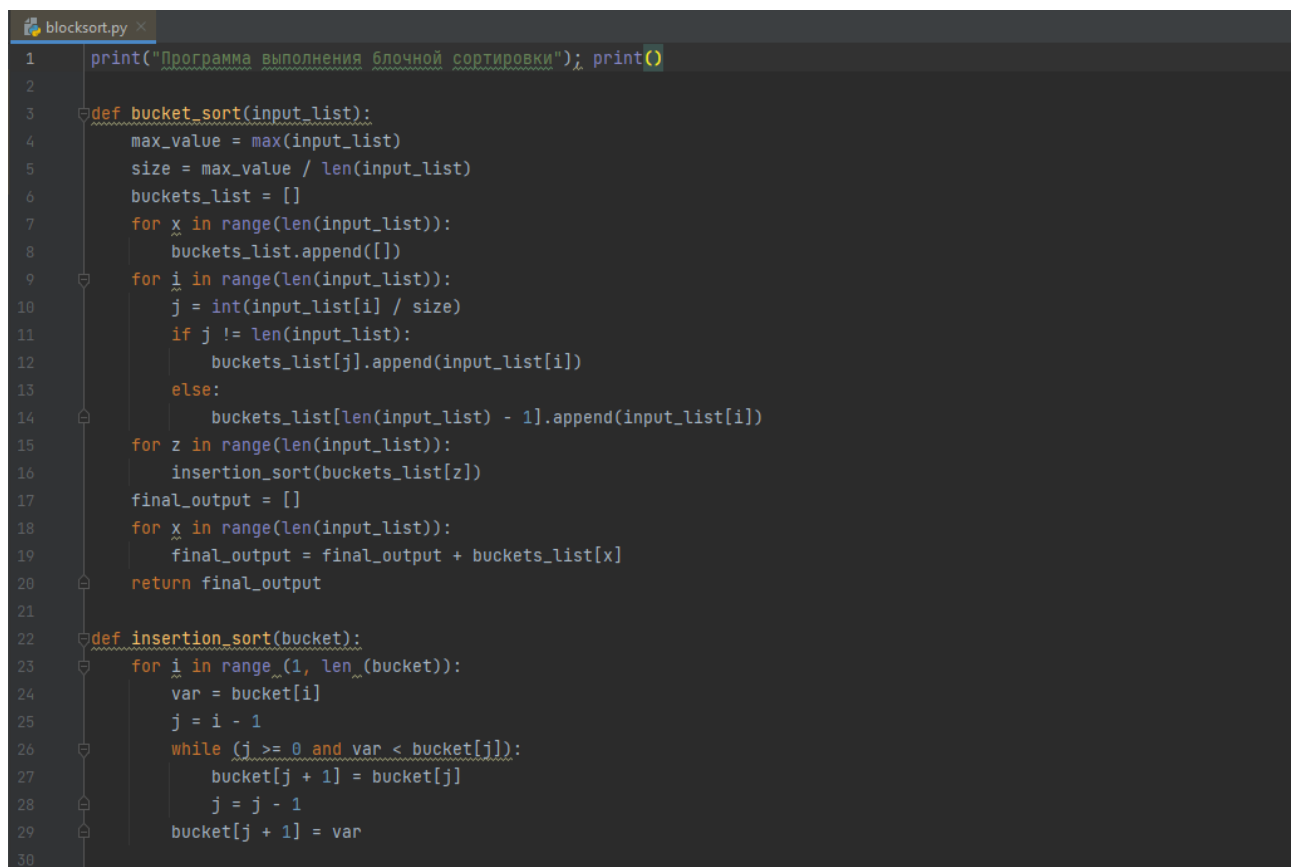
рис.2

## Задание 2

“Изучить блочную и пирамидальную сортировку. Написать соответствующие программы.”

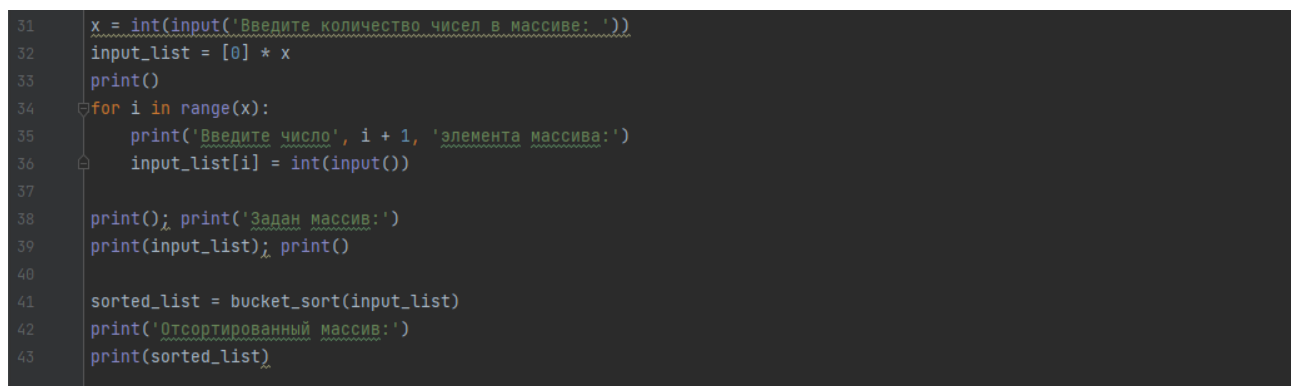
### Выполнение:

Пользуясь дополнительными источниками информации, мы изучили методы блочной и пирамидальной сортировок. После изучения мы написали соответствующие программы. Коды программ представлены ниже (рис.3-5) и добавлены в репозиторий (наименования файлов: “blocksort.py”, “piramidsort.py”).



```
1 print("Программа выполнения блочной сортировки"); print()
2
3 def bucket_sort(input_list):
4     max_value = max(input_list)
5     size = max_value / len(input_list)
6     buckets_list = []
7     for x in range(len(input_list)):
8         buckets_list.append([])
9     for i in range(len(input_list)):
10        j = int(input_list[i] / size)
11        if j != len(input_list):
12            buckets_list[j].append(input_list[i])
13        else:
14            buckets_list[len(input_list) - 1].append(input_list[i])
15    for z in range(len(input_list)):
16        insertion_sort(buckets_list[z])
17    final_output = []
18    for x in range(len(input_list)):
19        final_output = final_output + buckets_list[x]
20    return final_output
21
22 def insertion_sort(bucket):
23     for i in range(1, len(bucket)):
24         var = bucket[i]
25         j = i - 1
26         while (j >= 0 and var < bucket[j]):
27             bucket[j + 1] = bucket[j]
28             j = j - 1
29         bucket[j + 1] = var
30
```

рис.3



```
31 x = int(input('Введите количество чисел в массиве: '))
32 input_list = [0] * x
33 print()
34 for i in range(x):
35     print('Введите число', i + 1, 'элемента массива:')
36     input_list[i] = int(input())
37
38 print(); print('Задан массив:')
39 print(input_list); print()
40
41 sorted_list = bucket_sort(input_list)
42 print('Отсортированный массив:')
43 print(sorted_list)
```

рис.4

Данная программа запрашивает у пользователя размерность массива, а затем значение каждого элемента этого массива. На экран выводится заданный массив. Далее программа производит сортировку заданного массива блочным методом и выводит на экран уже отсортированный массив.

```

piramidsort.py x
1  print("Программа выполнения пирамидальной сортировки"); print()
2
3  x=int(input('Введите количество чисел в массиве: '))
4  a=[0]*x
5  print()
6  for i in range(x):
7      print('Введите число', i+1, 'элемента массива:')
8      a[i]=int(input())
9  print(); print('Задан массив:', a); print()
10
11 def heapify(array, n, i):
12     largest = i
13     l = 2 * i + 1
14     r = 2 * i + 2
15
16     if l < n and array[i] < array[l]:
17         largest = l
18     if r < n and array[largest] < array[r]:
19         largest = r
20
21     if largest != i:
22         array[i], array[largest] = array[largest], array[i]
23         heapify(array, n, largest)
24
25
26 def heapSort(array):
27     n = len(array)
28     c = 0
29     for i in range(n // 2, -1, -1):
30         heapify(array, n, i)
31     for i in range(n - 1, 0, -1):
32         array[i], array[0] = array[0], array[i]
33         heapify(array, i, 0)
34         c += 1
35         print("Действие №", c, array)
36     print()
37     return array
38
39 print('Отсортированный массив:', heapSort(a))

```

рис.5

Данная программа запрашивает у пользователя размерность массива, а затем значение каждого элемента этого массива. На экран выводится заданный массив. Далее программа производит сортировку массива пирамидальным методом и выводит на экран каждую стадию сортировки. В конечном итоге на экран выводится отсортированный массив.

### Задание 3

“Оцените достоинства, недостатки и сложность изученных методов сортировок.”

#### Выполнение:

##### 1. Быстрая сортировка:

###### а. Достоинства:

- i. Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
- ii. Требуется лишь  **$O(1)$**  дополнительной памяти для своей работы (неулучшенный рекурсивный алгоритм - в худшем случае  **$O(n)$**  памяти)
- iii. Хорошо сочетается с механизмами кэширования и виртуальной памяти.
- iv. Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах).
- v. Допускает эффективную модификацию для сортировки по нескольким ключам: благодаря тому, что в процессе разделения автоматически выделяется отрезок элементов, равных опорному, этот отрезок можно сразу же сортировать по следующему ключу.
- vi. Работает на связных списках и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

###### б. Недостатки:

- i. Сильно деградирует по скорости (до  **$O(n^2)$** ) в худшем или близком к нему случае, что может случиться при неудачных входных данных.
- ii. Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека, так как в худшем случае ей может потребоваться сделать  **$O(n)$**  вложенных рекурсивных вызовов.
- iii. Неустойчив.

###### в. Сложность по времени:

- i. Лучшее время:  **$O(n \log_2 n)$**
- ii. Среднее время:  **$O(n \log n)$**
- iii. Худшее время:  **$O(n^2)$**

**d. Сложность по памяти**

- i.  $O(n)$
- ii.  $O(\log n)$

**2. Сортировка расчёской:**

**a. Достоинства:**

- i. Достаточно простой алгоритм, построенный на основе пузырьковой сортировки

**b. Недостатки:**

- i. Скорость выполнения программы. Выполняется большое количество перестановок, на что тратится много процессорного времени.
- ii. Производит полный перебор всех элементов массива.

**c. Сложность по времени:**

- i. Лучшее время:  $O(n \log n)$
- ii. Среднее время:  $O(n^2 / 2^P)$
- iii. Худшее время:  $O(n^2)$

**d. Сложность по памяти:**

- i.  $O(n)$
- ii.  $O(1)$

**3. Блочная сортировка:**

**a. Достоинства:**

- i. Относится к классу быстрых алгоритмов с линейным временем исполнения  $O(N)$  (на удачных входных данных).

**b. Недостатки:**

- i. Сильно деградирует при большом количестве мало отличных элементов, или же на неудачной функции получения номера корзины по содержимому элемента.

**c. Сложность по времени:**

- i.  $O(n)$



#### 4. Пирамидальная сортировка:

##### a. Достоинства:

- i. Имеет доказанную оценку худшего случая  $O(n \log n)$
- ii. Сортирует на месте, то есть требует всего  $O(1)$  дополнительной памяти.

##### b. Недостатки:

- i. Неустойчив — для обеспечения устойчивости нужно расширять ключ.
- ii. На почти отсортированных массивах работает столь же долго, как и на хаотических данных.
- iii. На одном шаге выборку приходится делать хаотично по всей длине массива — поэтому алгоритм плохо сочетается с кэшированием и подкачкой памяти.
- iv. Методу требуется «мгновенный» прямой доступ; не работает на связанных списках и других структурах памяти последовательного доступа.
- v. Не распараллеливается.

##### c. Сложность по времени:

- i.  $O(n \log n)$

##### d. Сложность по памяти:

- i.  $O(1)$