

Министерство образования и науки РФ
Национальный исследовательский университет «Высшая школа экономики»
Московский институт электроники и математики имени А.Н.Тихонова



NATIONAL RESEARCH
UNIVERSITY

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Методы программирования»
направления 10.05.01 «Компьютерная безопасность»

«Сортировки»
«Вариант 11»

Выполнила:
КУЗЬМИНА КСЕНИЯ
Студентка группы СКБ201

Москва, 2023

1 Задание

Массив данных о пассажирах некоторой авиакомпании: номер рейса, дата рейса, ФИО пассажира, номер места в самолете

Сравнение по полям – дата рейса, номер рейса, ФИО, номер места)

Сортировки: Пузырьком, Простыми вставками, Пирамидальная

2 Цель лабораторной работы

Познакомиться с различными алгоритмами сортировок, протестировать их скорость на сгенерированных данных и сделать выводы.

3 Сравнение скоростей сортировок

Размер данных	Пузырьковая сортировка	Сортировка вставками	Пирамидальная сортировка
100	0.005279064178466797	0.006344795227050781	0.009969711303710938
500	0.1503586769104004	0.08323192596435547	0.009595155715942383
1000	0.9631109237670898	0.43497610092163086	0.021537065505981445
2500	4.099838972091675	2.232435941696167	0.0684061050415039
5000	14.722675800323486	9.051411151885986	0.15944910049438477
7500	38.848604917526245	21.48124599456787	0.2532949447631836
10000	62.81747794151306	37.26092600822449	0.32654905319213867

Далее рассмотрим отдельные изображения скорости сортировок: Наконец, рассмотрим

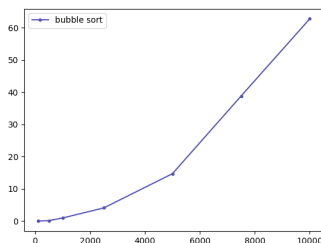


Рис. 1: Сортировка пузырьком

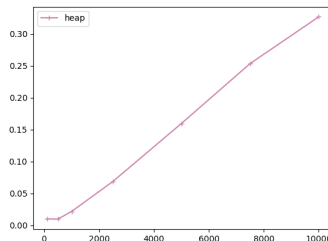


Рис. 2: Пирамидальная сортировка

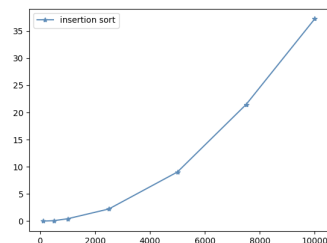
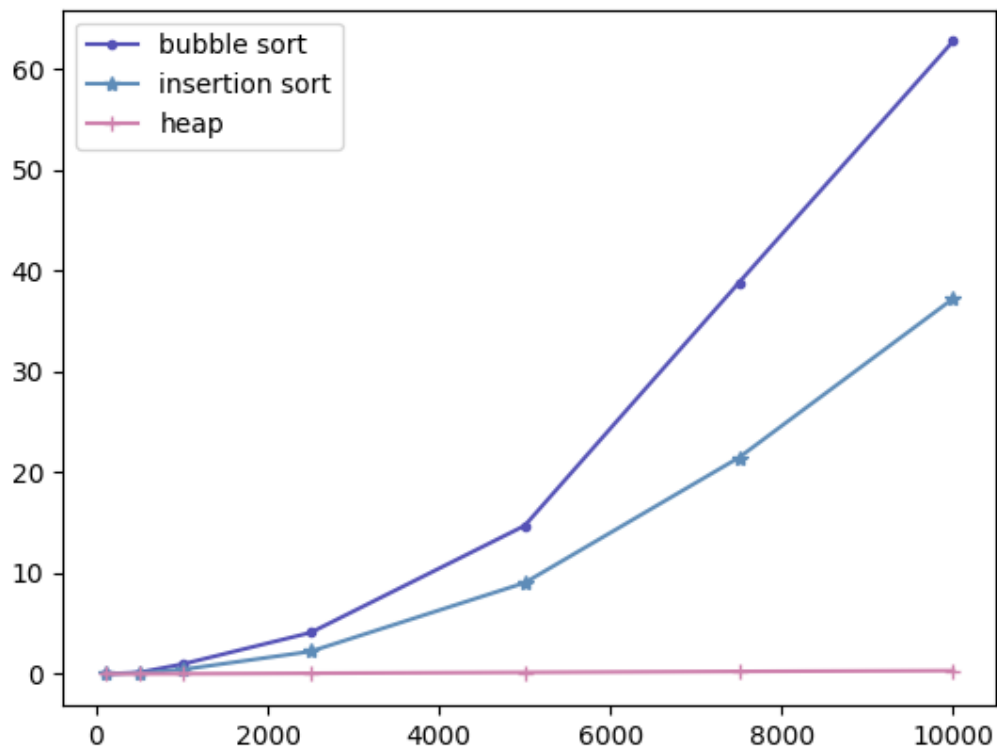


Рис. 3: Сортировка вставками

рисуем общий график для всех трёх сортировок, чтобы сравнить их скорости:



Общий график для трёх сортировок

4 Код программы

Ссылка на git-репозиторий: [source-code](#)

Файл **generator.py** создаёт таблицы со случайными данными: датой, номером рейса, ФИО пассажира и их местом в самолете.

```

1 from russian_names import RussianNames
2 import pandas as pd
3 import random
4 import time
5 import datetime
6
7 long_months = [1, 3, 5, 7, 8, 10, 12]
8 short_months = [4, 6, 9, 10]
9
10 def random_date(start, end, prop):
11     time_format = '%m/%d/%Y %I:%M %p'
12     ptime = time.mktime(time.strptime(start, time_format)) + prop * time.mktime(time.
13         strptime(end, time_format))
14     return time.strftime(time_format, time.localtime(ptime))
15
16 def line_generator():
17     arr = []
18     date = []
19     date.append(random.randint(2000, 2023))
20     date.append(random.randint(1, 12))
21     if date[-1] in long_months:
22         date.append(random.randint(1, 31))
23     elif date[-1] in short_months:

```

```

23     date.append(random.randint(1, 30))
24 else:
25     date.append(random.randint(1, 28))
26 date.append(random.randint(0, 23))
27 date.append(random.randint(0, 59))
28 date.append(random.randint(0, 59))
29 arr.append(datetime.datetime(date[0], date[1], date[2], date[3], date[4], date[5]))
30 arr.append(random.randint(0, 10000))
31 rn = RussianNames(count = 1, patronymic = True, transliterate = True)
32 for person in rn:
33     arr.append(person)
34 arr.append(random.randint(0,150))
35 return arr
36
37 def table_generator (n: int):
38     arr = []
39     for i in range(n):
40         arr.append(line_generator())
41     return arr
42
43 def xlsx_file_generator(n: int) -> object:
44     start_time = time.time()
45     df = pd.DataFrame(table_generator(n), columns = ['Date', 'Flight', 'Full Name', 'Place
46         '])
47     data_file = 'flight_info_' + str(n) + '.xlsx'
48     writer = pd.ExcelWriter(data_file, engine = 'xlsxwriter')
49     df.to_excel(writer, 'flight_table1')
50     writer.save()
51     print(data_file + ':')
52     print("          %s seconds          " % (time.time() - start_time))
53
54 xlsx_file_generator(100)
55 xlsx_file_generator(500)
56 xlsx_file_generator(1000)
57 xlsx_file_generator(2500)
58 xlsx_file_generator(5000)
59 xlsx_file_generator(7500)
60 xlsx_file_generator(10000)

```

Файл Data.py задаёт класс с этой информацией

```

1 import datetime
2
3 class Data:
4     def __init__(self, Date: datetime.datetime, Flight: int, FullName: str, Place: int):
5         self.FullName = FullName
6         self.Flight = Flight
7         self.Date = Date
8         self.Place = Place
9
10    def __eq__(self, other):
11        return self.FullName == other.FullName and self.Flight == other.Flight and \
12            self.Date == other.Date and self.Place == other.Place
13
14    def __gt__(self, other): # >
15        if self == other:
16            return True
17        if self.Date == other.Date:
18            if self.Flight == other.Flight:
19                if self.FullName == other.FullName:
20                    return self.Place > other.Place
21                else:
22                    return self.FullName < other.FullName
23            else:
24                return self.Flight > other.Flight
25        else:
26            return self.Date > other.Date
27
28    def __lt__(self, other):
29        return not self > other

```

```

30
31     def __ge__(self, other):
32         return self > other or self == other
33
34     def __le__(self, other):
35         return other > self or self == other
36
37     def __repr__(self):
38         string = str(self.Flight) + " " + str(self.Date) + " " + self.FullName + " " +
39         str(self.Place)
40         return string
41
42     def __str__(self):
43         string = str(self.Flight) + " " + str(self.Date) + " " + self.FullName + " " +
44         str(self.Place)
45         return string

```

Файл **main.py** - функции сортировок и обработки информации из excel файлов

```

1  import time
2  import pandas as pd
3  import generator
4  import Data
5
6  def parent(i):
7      return (i - 1) // 2
8
9  def left(i):
10     return 2 * i + 1
11
12  def right(i):
13     return 2 * i + 2
14
15  def to_xls_file(arr, file_name: str):
16     df = pd.DataFrame(data = data_array(arr), columns = ['Date', 'Flight', 'Full Name',
17     'Place'])
18     file_name = file_name + '.xlsx'
19     writer = pd.ExcelWriter(file_name, engine = 'xlsxwriter')
20     df.to_excel(writer, 'flight_table1')
21     writer.save()
22
23  def from_xlsx_file(file_name: str):
24     xl = pd.ExcelFile(file_name)
25     df = xl.parse('flight_table1')
26     arr = []
27     for i in range(len(df)):
28         row = df.iloc[i]
29         data = Data.Data(row[1].to_pydatetime(), int(row[2]), str(row[3]), int(row[4]))
30         arr.append(data)
31     return arr
32
33  def data_array(array):
34     arr = []
35     for i in range(len(array)):
36         arr2 = []
37         arr2.append(array[i].Date)
38         arr2.append(array[i].Flight)
39         arr2.append(array[i].FullName)
40         arr2.append(array[i].Place)
41         arr.append(arr2)
42     return arr
43
44  def insertion_sort(data_file: str, text_file: str):
45     f = open(text_file, 'a')
46     arr = from_xlsx_file(data_file)
47     start_time = time.time()
48     for i in range(1, len(arr)):
49         j = i
50         temp = arr[j]
51         while j > 0 and temp < arr[j - 1]:

```

```

51         arr[j] = arr[j - 1]
52         j = j - 1
53         arr[j] = temp
54     print(data_file + ' :')
55     print("          %s seconds          " % (time.time() - start_time))
56     note = data_file + ' time = ' + str(time.time() - start_time) + ' !! INSERT SORT !!'
57     f.write(note)
58     f.close()
59     note2 = data_file.rsplit(".", 1)[0] + '_insert_sorted'
60     to_xls_file(arr, note2)
61
62 def heapify(heap, index, size):
63     l = left(index)
64     r = right(index)
65     if (l < size and heap[l] > heap[index]):
66         largest = l
67     else:
68         largest = index
69     if (r < size and heap[r] > heap[largest]):
70         largest = r
71     if (largest != index):
72         heap[largest], heap[index] = heap[index], heap[largest]
73         heapify(heap, largest, size)
74
75
76 def heap_sort(data_file: str, text_file: str):
77     f = open(text_file, 'a')
78     array = from_xlsx_file(data_file)
79     start_time = time.time()
80     length = len(array)
81     beginning = parent(length - 1)
82     while beginning >= 0:
83         heapify(array, index=beginning, size=length)
84         beginning = beginning - 1
85     for i in range(len(array) - 1, 0, -1):
86         array[0], array[i] = array[i], array[0]
87         heapify(array, index=0, size=i)
88     print(data_file + ' :')
89     print("          %s seconds          " % (time.time() - start_time))
90     note = data_file + ' time = ' + str(time.time() - start_time) + ' !! HEAP SORT !!'
91     f.write(note)
92     f.close()
93     note2 = data_file.rsplit(".", 1)[0] + '_heap_sorted'
94     to_xls_file(array, note2)
95
96 def bubble_sort(data_file: str, text_file: str):
97     f = open(text_file, 'a')
98     array = from_xlsx_file(data_file)
99     start_time = time.time()
100    for i in range(len(array) - 1):
101        for j in range(len(array) - i - 1):
102            if array[j] > array[j + 1]:
103                buff = array[j]
104                array[j] = array[j + 1]
105                array[j + 1] = buff
106    print(data_file + ' :')
107    print("          %s seconds          " % (time.time() - start_time))
108    note = data_file + ' time = ' + str(time.time() - start_time) + ' !! BUBBLE SORT !!'
109    f.write(note)
110    f.close()
111    note2 = data_file.rsplit(".", 1)[0] + '_bubblesort'
112    to_xls_file(array, note2)
113
114 bubble_sort('flight_info_100.xlsx', 'times.txt')
115 insertation_sort('flight_info_100.xlsx', 'times.txt')
116 heap_sort('flight_info_100.xlsx', 'times.txt')
117

```

```

118 bubble_sort('flight_info_500.xlsx', 'times.txt')
119 insertation_sort('flight_info_500.xlsx', 'times.txt')
120 heap_sort('flight_info_500.xlsx', 'times.txt')
121
122 bubble_sort('flight_info_1000.xlsx', 'times.txt')
123 insertation_sort('flight_info_1000.xlsx', 'times.txt')
124 heap_sort('flight_info_1000.xlsx', 'times.txt')
125
126 bubble_sort('flight_info_2500.xlsx', 'times.txt')
127 insertation_sort('flight_info_2500.xlsx', 'times.txt')
128 heap_sort('flight_info_2500.xlsx', 'times.txt')
129
130 bubble_sort('flight_info_5000.xlsx', 'times.txt')
131 insertation_sort('flight_info_5000.xlsx', 'times.txt')
132 heap_sort('flight_info_5000.xlsx', 'times.txt')
133
134 bubble_sort('flight_info_7500.xlsx', 'times.txt')
135 insertation_sort('flight_info_7500.xlsx', 'times.txt')
136 heap_sort('flight_info_7500.xlsx', 'times.txt')
137
138 bubble_sort('flight_info_10000.xlsx', 'times.txt')
139 insertation_sort('flight_info_10000.xlsx', 'times.txt')
140 heap_sort('flight_info_10000.xlsx', 'times.txt')

```

5 Вывод

По результатам работы видно, что на больших числах пузырьковая сортировка и сортировка вставками намного медленнее, чем пирамидальная сортировка. Это подтверждают и теоритические данные – вычислительная сложность первых двух $O(n^2)$, а последней – $O(n \cdot \log n)$