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

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, рассмот-

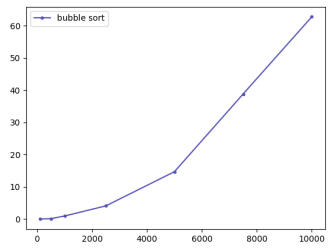


Рис. 1: Сортировка пузырьком

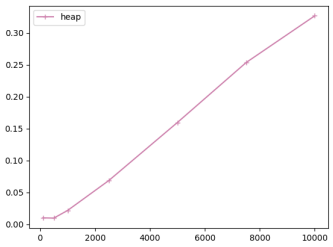


Рис. 2: Пирамидальная сортировка

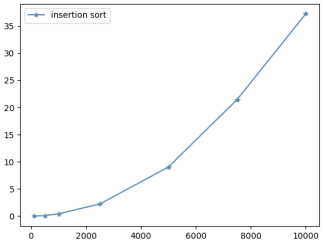
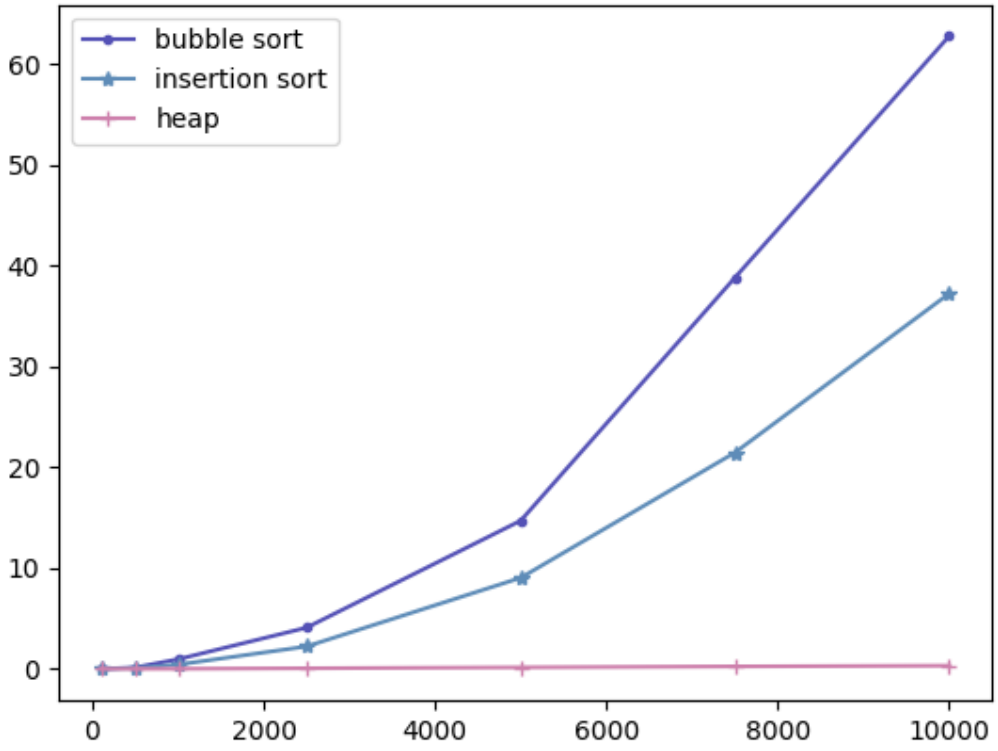


Рис. 3: Сортировка вставками

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



Общий график для трёх сортировок

4 Код программы

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

```
from russian_names import RussianNames
import pandas as pd
import random
import time
import datetime

long_months = [1, 3, 5, 7, 8, 10, 12]
short_months = [4, 6, 9, 10]

def random_date(start, end, prop):
    time_format = '%m/%d/%Y %I:%M %p'
    ptime = time.mktime(time.strptime(start, time_format)) + prop * time.mktime(time.
    strptime(end, time_format))
    return time.strftime(time_format, time.localtime(ptime))

def line_generator():
  arr = []
  date = []
  date.append(random.randint(2000, 2023))
  date.append(random.randint(1, 12))
  if date[-1] in long_months:
      date.append(random.randint(1, 31))
  elif date[-1] in short_months:
      date.append(random.randint(1, 30))
```

```
  else:
      date.append(random.randint(1, 28))
  date.append(random.randint(0, 23))
  date.append(random.randint(0, 59))
  date.append(random.randint(0, 59))
  arr.append(datetime.datetime(date[0], date[1], date[2], date[3], date[4], date[5]))
  arr.append(random.randint(0, 10000))
  rn = RussianNames(count = 1, patronymic = True, transliterate = True)
  for person in rn:
        arr.append(person)
  arr.append(random.randint(0,150))
  return arr

def table_generator (n: int):
  arr = []
  for i in range(n):
    arr.append(line_generator())
  return arr

def xlsx_file_generator(n: int) -> object:
  start_time = time.time()
  df = pd.DataFrame(table_generator(n), columns = ['Date', 'Flight', 'Full Name', 'Place
    '])
  data_file = 'flight_info_' + str(n) + '.xlsx'
  writer = pd.ExcelWriter(data_file, engine ='xlsxwriter')
  df.to_excel(writer, 'flight_table1')
  writer.save()
  print(data_file + ':')
  print("       %s seconds       " % (time.time() - start_time))

xlsx_file_generator(100)
xlsx_file_generator(500)
xlsx_file_generator(1000)
xlsx_file_generator(2500)
xlsx_file_generator(5000)
xlsx_file_generator(7500)
xlsx_file_generator(10000)
```

Файл **Data.py** задаёт класс с этой информацией

```
import datetime

class Data:
    def __init__(self, Date: datetime.datetime, Flight: int, FullName: str, Place: int):
        self.FullName = FullName
        self.Flight = Flight
        self.Date = Date
        self.Place = Place

    def __eq__(self, other):
        return self.FullName == other.FullName and self.Flight == other.Flight and \
               self.Date == other.Date and self.Place == other.Place

    def __gt__(self, other):  # >
        if self == other:
            return True
        if self.Date == other.Date:
            if self.Flight == other.Flight:
                if self.FullName == other.FullName:
                    return self.Place > other.Place
                else:
                    return self.FullName < other.FullName
            else:
                return self.Flight > other.Flight
        else:
            return self.Date > other.Date

    def __lt__(self, other):
        return not self > other

```

```
    def __ge__(self, other):
        return self > other or self == other

    def __le__(self, other):
        return other > self or self == other

    def __repr__(self):
        string = str(self.Flight) + " " + str(self.Date) + " " + self.FullName + " " + str(self.Place)
        return string

    def __str__(self):
        string = str(self.Flight) + " " + str(self.Date) + " " + self.FullName + " " + str(self.Place)
        return string
```

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

```
import time
import pandas as pd
import generator
import Data

def parent(i):
    return (i - 1) // 2

def left(i):
    return 2 * i + 1

def right(i):
    return 2 * i + 2

def to_xls_file(arr, file_name: str):
    df = pd.DataFrame(data = data_array(arr), columns = ['Date', 'Flight', 'Full Name', 'Place'])
    file_name = file_name + '.xlsx'
    writer = pd.ExcelWriter(file_name, engine ='xlsxwriter')
    df.to_excel(writer, 'flight_table1')
    writer.save()

def from_xlsx_file(file_name: str):
    xl = pd.ExcelFile(file_name)
    df = xl.parse('flight_table1')
    arr = []
    for i in range(len(df)):
        row = df.iloc[i]
        data = Data.Data(row[1].to_pydatetime(), int(row[2]), str(row[3]), int(row[4]))
        arr.append(data)
    return arr

def data_array(array):
    arr = []
    for i in range(len(array)):
        arr2 = []
        arr2.append(array[i].Date)
        arr2.append(array[i].Flight)
        arr2.append(array[i].FullName)
        arr2.append(array[i].Place)
        arr.append(arr2)
    return arr

def insertation_sort(data_file: str, text_file: str):
    f = open(text_file, 'a')
    arr = from_xlsx_file(data_file)
    start_time = time.time()
    for i in range(1, len(arr)):
        j = i
        temp = arr[j]
        while j > 0 and temp < arr[j - 1]:
            arr[j] = arr[j - 1]
```

```
            j = j - 1
        arr[j] = temp
    print(data_file + ' :')
    print("       %s seconds       " % (time.time() - start_time))
    note = data_file + ' time = ' + str(time.time() - start_time) + '  !! INSERT SORT !!
    ' + '\n'
    f.write(note)
    f.close()
    note2 = data_file.rsplit(".", 1)[0] + '_insert_sorted'
    to_xls_file(arr, note2)

def heapify(heap, index, size):
    l = left(index)
    r = right(index)
    if (l < size and heap[l] > heap[index]):
        largest = l
    else:
        largest = index
    if (r < size and heap[r] > heap[largest]):
        largest = r
    if (largest != index):
        heap[largest], heap[index] = heap[index], heap[largest]
        heapify(heap, largest, size)


def heap_sort(data_file: str, text_file: str):
    f = open(text_file, 'a')
    array = from_xlsx_file(data_file)
    start_time = time.time()
    length = len(array)
    beginning = parent(length - 1)
    while beginning >= 0:
        heapify(array, index=beginning, size=length)
        beginning = beginning - 1
    for i in range(len(array) - 1, 0, -1):
        array[0], array[i] = array[i], array[0]
        heapify(array, index=0, size=i)
    print(data_file + ' :')
    print("       %s seconds       " % (time.time() - start_time))
    note = data_file + ' time = ' + str(time.time() - start_time) + '  !! HEAP SORT !!'
    + '\n'
    f.write(note)
    f.close()
    note2 = data_file.rsplit(".", 1)[0] + '_heap_sorted'
    to_xls_file(array, note2)

def bubble_sort(data_file: str, text_file: str):
    f = open(text_file, 'a')
    array = from_xlsx_file(data_file)
    start_time = time.time()
    for i in range(len(array) - 1):
        for j in range(len(array) - i - 1):
            if array[j] > array[j + 1]:
                buff = array[j]
                array[j] = array[j + 1]
                array[j + 1] = buff
    print(data_file + ' :')
    print("       %s seconds       " % (time.time() - start_time))
    note = data_file + ' time = ' + str(time.time() - start_time) + ' !! BUBBLE SORT !!'
     + '\n'
    f.write(note)
    f.close()
    note2 = data_file.rsplit(".", 1)[0] + '_bubblesort'
    to_xls_file(array, note2)

bubble_sort('flight_info_100.xlsx', 'times.txt')
insertation_sort('flight_info_100.xlsx', 'times.txt')
heap_sort('flight_info_100.xlsx', 'times.txt')

bubble_sort('flight_info_500.xlsx', 'times.txt')
```

```
insertation_sort('flight_info_500.xlsx', 'times.txt')
heap_sort('flight_info_500.xlsx', 'times.txt')

bubble_sort('flight_info_1000.xlsx', 'times.txt')
insertation_sort('flight_info_1000.xlsx', 'times.txt')
heap_sort('flight_info_1000.xlsx', 'times.txt')

bubble_sort('flight_info_2500.xlsx', 'times.txt')
insertation_sort('flight_info_2500.xlsx', 'times.txt')
heap_sort('flight_info_2500.xlsx', 'times.txt')

bubble_sort('flight_info_5000.xlsx', 'times.txt')
insertation_sort('flight_info_5000.xlsx', 'times.txt')
heap_sort('flight_info_5000.xlsx', 'times.txt')

bubble_sort('flight_info_7500.xlsx', 'times.txt')
insertation_sort('flight_info_7500.xlsx', 'times.txt')
heap_sort('flight_info_7500.xlsx', 'times.txt')

bubble_sort('flight_info_10000.xlsx', 'times.txt')
insertation_sort('flight_info_10000.xlsx', 'times.txt')
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 – $O(n \cdot \log n)$