**Правительство Российской Федерации**

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Кафедра «Компьютерная безопасность»

**ОТЧЁТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №1**

**по дисциплине**

**«Методы программирования»**

Работу выполнил студент группы СКБ-192

Подпись, дата

Ташкинов С.О.

Работу проверил

Подпись, дата

Драчев Г.А.

Москва 2022

**Постановка задачи**

1. Реализовать на языке С++ сортировки для массива объектов в соответствии с вариантом.
2. Перегрузить операторы сравнения (>, <, >=, <=) для сравнения объектов.
3. Входные данные для сортировки массива обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).
4. Выбрать 7-10 наборов данных для сортировки размерности от 100 и более (но не менее 100000). Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Сделать вывод о том, в каком случае, какой из методов лучше применять. Графики можно строить программно или в любой из прикладных программ (MS Excel, Matlab, MathCad и т.д.).
5. Сделать отчет, содержащий титульный лист, код программы со спецификациями каждого метода и подробными комментариями, графики скоростей сортировок и выводы.

Вариант 21:

Массив данных об авиарейсах, прибывающих в аэропорт: Номер рейса, название авиакомпании, дата прилета, время прилета по расписанию, число пассажиров на борту (сравнение по полям – дата прилета, время прилета, название авиакомпании, число пассажиров (по убыванию))

1. Сортировка пузырьком
2. Шейкер сортировка
3. Быстрая сортировка

**1. Алгоритм решения задачи**

Данная лабораторная работа была реализована на языке программирования Python 3.9.1. Графики были построены с помощью Wolfram Mathematica. В файле Lab1.py определен класс FlightsArr, его конструктор и перегрузки операторов, а также функции реализующие сортировку пузырьком - bubble\_sort(arr: list), шейкер сортировку - cocktail\_shaker\_sort(arr: list), быструю сортировку - quick\_sort(arr: list). В файле Generator.py происходит генерация данных с помощью функции gen(n) и запись сгенерированных данных в файл db\_{n}.txt.

**2.Выполнение задания**

**2.1. Функция bubble\_sort(arr: list) -> list.**

Функция **bubble\_sort**(arr: list)сортирует переданный массив, используя алгоритм сортировки пузырьком. Проходим массив от начала до конца массива N (длинна массива) раз, меняя соседние элементы местами, если последующее меньше предыдущего.

**2.2. Функции** **cocktail\_shaker\_sort (arr: list) -> list.**

Функция cocktail\_shaker\_sort(arr: list) сортирует переданный массив, использую алгоритм шейкер сортировки. Алгоритм похож на сортировку пузырьком, кроме того, что мы поочередно сортируем слева-направо и уменьшаем область сортировки на один каждый раз.

**2.3. Функции quick\_sort (arr: list) -> list.**

Функция **quick\_sort** (arr: list) сортирует переданный массив, используя алгоритм быстрой сортировки. Выбираем случайный элемент массива, после этого делим массив на два массива, левый (элементы массива меньшие выбранного) и правый (элементы массива больше правого). Рекурсивно выполняем функцию для левого и правого массива. Если размер массива, передаваемого в функцию меньше или равен единицы, возвращаем исходный массив.

**2.4. Класс FlightsArr.**

Класс FlightsArr содержит в себе информацию обо всех прибывающий рейсах

id: int – номер рейса

name: str – название компании

date: datetime.date – дата прилета,

time: datetime.time – время прилета.

numbers: int – число пассажиров

Функция \_\_init\_\_(self, id, name, date, time, number):– конструктор и инициализирует вышеперечисленные атрибуты класса переданными значениями.

В функциях \_\_lt\_\_(self, other), \_\_gt\_\_(self, other), \_\_le\_\_(self, other), \_\_ge\_\_(self, other)реализованы перегрузки операторов <, >, <= и >= соответственно. Сравнение проводится по полям в следующем приоритете: дата прилета, время прилета, название авиакомпании, число пассажиров.

**2.5. Функция gen(n)**

Генерирует случайный список данных для создания объектов класса FlightsArr. Для генерации используется модуль Faker.

**3. Построение графиков и выводы**

В результате работы программы были получены следующие данные

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов | Время сортировки пузырьком, м:с | Время шейкер сортировки, м:c | Время быстрой сортировки, мс |
| 100 | 0:0.003989 | 0:00.002054 | 1.185 |
| 250 | 0:0.023836 | 0:0.012793 | 2.026 |
| 500 | 0:0.097848 | 0:0.055858 | 5.998 |
| 750 | 0:0.225437 | 0:0.128878 | 6.850 |
| 1000 | 0:0.505061 | 0:0.261860 | 9.973 |
| 2500 | 0:2.917664 | 0:1.505166 | 27.920 |
| 5000 | 0:12.682606 | 0:7.396378 | 80.908 |
| 7500 | 0:25.930941 | 0:15.896633 | 99.524 |
| 10000 | 0:46.579348 | 0:25.186984 | 127.640 |
| 25000 | 5:18.124050 | 02:52.854926 | 364.031 |
| 50000 | 26:15.314492 | 13:02.585328 | 1248.311 |
| 100000 | 103:34.056987 | 52:20.789101 | 3052.129 |

Таблица 1 - результаты работы программы

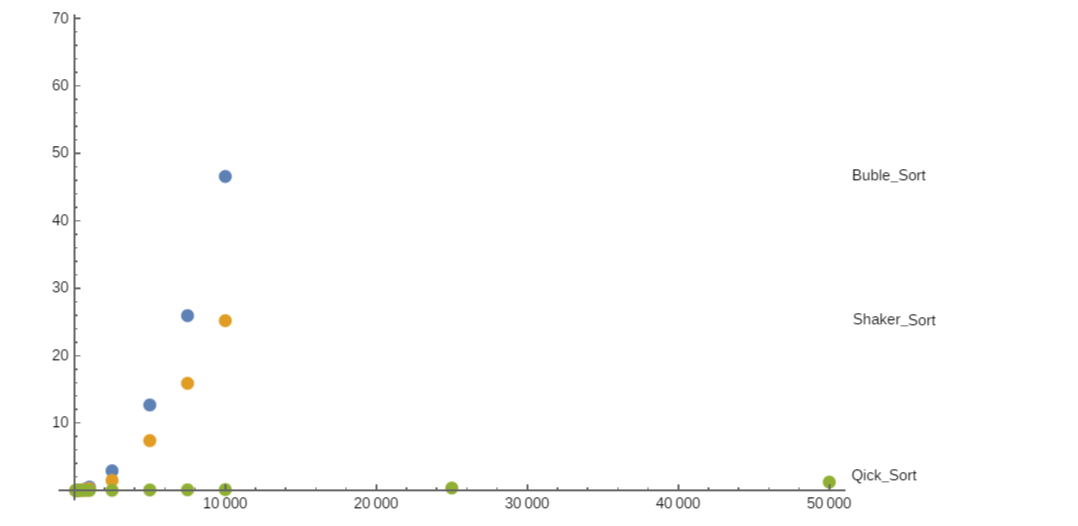


Рисунок 1 - результаты работы программы

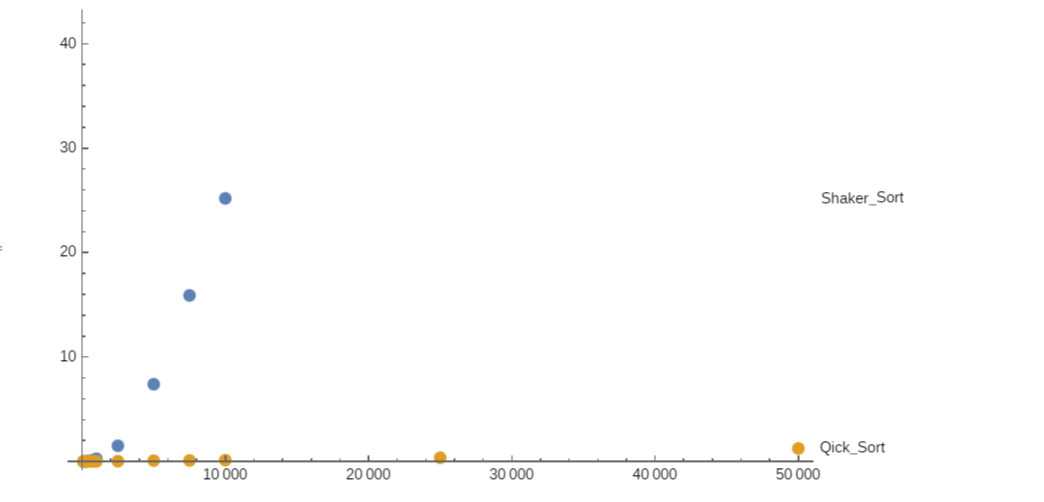


Рисунок 2 - результаты работы программы

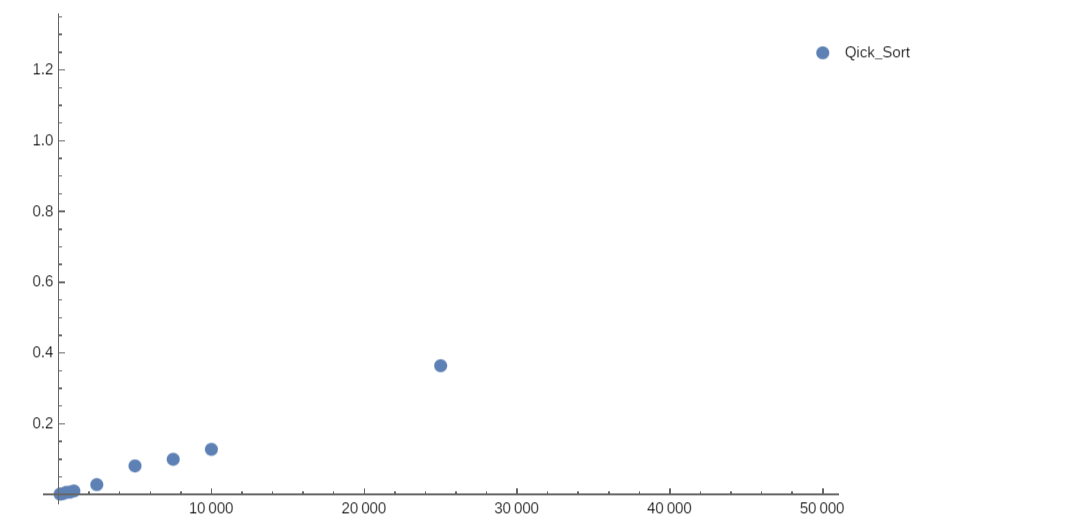


Рисунок 3 - результаты работы программы

Временная теоретическая сложность реализованных алгоритмов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сортировка пузырьком | Шейкер сортировка | Быстрая сортировка |
| Худший случай | O(n2) | O(n2) | O(n2) |
| Средний случай | O(n2) | O(n2) | O(n\*logn) |
| Лучший случай | O(n) | O(n) | O(n\*logn) |

Таблица 2 - сложности алгоритмов

Как мы видим из таблицы, теоретические сложности всех трех алгоритмов согласуются с полученными графиками. Из полученных данных можно сделать вывод, что для сортировки данных лучше всего использовать быструю сортировку, при больших размерах массивов, быстрая сортировка работает в разы быстрее, чем две другие сортировки. Шейк сортировка в среднем в два раза быстрее, чем сортировка пузырьком.

**Приложение А**

A1. Исходный код файла **Lab1.py**

import datetime

import random

class FlightsArr:

def \_\_init\_\_(self, id, name, date, time, number):

"""

Конструктор класса Flights\_Arr

:param id:

:param name:

:param date:

:param time:

:param number:

"""

self.id = id

self.name = name

self.date = date

self.time = time

self.number = number

def \_\_lt\_\_(self, other):

"""

Перегрузка оператора меньше

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект меньше объекта other, иначе False

:rtype: bool

"""

if self.date < other.date:

return True

elif self.date == other.date:

if self.time < other.time:

return True

elif self.time == other.time:

if self.name < other.name:

return True

elif self.name == other.name:

if self.number > other.number:

return True

return False

def \_\_le\_\_(self, other):

"""

Перегрузка оператора меньше или равно

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект меньше или равен объекта other, иначе False

:rtype: bool

"""

return self.\_\_lt\_\_(other) or (self.date == other.date and

self.time == other.time and

self.name == other.name and

self.number == other.number)

def \_\_gt\_\_(self, other):

"""

Перегрузка оператора больше

:param other: Объект класса :class: `FlightsArr`, с которым проводится сравнение

:type other: :class: `FlightsArr`

:return: True, если данный объект больше объекта other, иначе False

:rtype: bool

"""

return not self.\_\_le\_\_(other)

def \_\_ge\_\_(self, other):

"""

Перегрузка оператора больше или равно

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект больше или равен объекта other, иначе False

:rtype: bool

"""

return not self.\_\_lt\_\_(other)

def \_\_str\_\_(self):

"""

Строковое представление объекта

:return: строковое представление

:rtype: str

"""

return str(self.id) + " " + self.name + " " + str(self.date) + " " + str(self.time) + " " + str(self.number)

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

def bubble\_sort(arr: list) -> list:

"""

Сортировка пузырьком

:param arr: Массив для сортировки

:type arr: list

:return arr: Отсортированный массив

:rtype: list

"""

N = len(arr)-1 # индекс последнего элемента

for i in range(N): # идём от начала до конца массива

for j in range(N): # проходим от начала до конца массива N раз

if(arr[j] > arr[j+1]): # если j элемент массива больше j+1

arr[j], arr[j + 1] = arr[j + 1], arr[j] # меняем значение jго элемена на j+1-ый и наоборот

return arr # возвращаем отсортированный массив

def cocktail\_shaker\_sort(arr: list) -> list:

"""

Шейкер сортировка

:param arr: Массив для сортировки

:type arr: list

:return: Отсортированный массив

:rtype: list

"""

left = 0 # индекс крайнего-левого элемента массива

right = len(arr) - 1 # индекс крайнего-правого элемента массива

while left <= right: # пока индекс первого элемента меньше последнего

for i in range(left, right, +1): # идем от left до right с шагом один

if arr[i] > arr[i + 1]: # если i элемент больше i+1-го

arr[i], arr[i + 1] = arr[i + 1], arr[i] # меняем местами i и i+1-ый элементы

right -= 1 # сдвигаем индекс крайнего-првого элемента влево

for i in range(right, left, -1): # идем от right до left с шагом один

if arr[i - 1] > arr[i]: # если i-1 ый элемент больше i

arr[i], arr[i - 1] = arr[i - 1], arr[i] # меняем местами i и i-1 ый элементы

left += 1 # сдвигаем индекс крайнего-левого элемента вправо

return arr # возвращаем отсортированный массив

def quick\_sort(arr: list) -> list:

"""

Быстрая сортировка

:param arr: Массив для сортировки

:type arr: list

:return: Отсортированный массив

:rtype: list

"""

if len(arr) <= 1: # если длина массива меньше и равна 1

return arr # возвращаем массив

else: # иначе

q = random.choice(arr) # выбираем случайное значение в массиве

l\_arr = [n for n in arr if n < q] # составляем массив из элементов меньше фиксированного

e\_arr = [q] \* arr.count(q) # составляем массив из всех элементов принимащих фиксированное значение

r\_arr = [n for n in arr if n > q] # составляем массив из элементов больше фиксированного

return quick\_sort(l\_arr) + e\_arr + quick\_sort(r\_arr) # рекрсивно выполнеяем программу для левого и правого массива

if \_\_name\_\_ == '\_\_main\_\_':

for n in [100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000]:

arr = []

with open(f"db\_{n}.txt", "r") as db:

data = db.read()

data\_lst = data.split("\n")

for i in data\_lst[:-1]:

per = i.split("--")

per[-3] = datetime.date.fromisoformat(per[-3])

per[-2] = datetime.time.fromisoformat(per[-2])

per[-1] = int(per[-1])

arr.append(FlightsArr(\*per))

print(f"{n}:")

start\_time = datetime.datetime.now()

arr\_bubble = bubble\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Сортировка пузырьком {end\_time - start\_time}")

with open(f"sorted\_bubble{n}.txt", "w") as f:

f.write(str(arr\_bubble))

start\_time = datetime.datetime.now()

arr\_shake = cocktail\_shaker\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Шейкер сортировка {end\_time - start\_time}")

with open(f"sorted\_shake{n}.txt", "w") as f:

f.write(str(arr\_shake))

start\_time = datetime.datetime.now()

arr\_quick = quick\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Быстрая сортировка {end\_time - start\_time}")

with open(f"sorted\_quick{n}.txt", "w") as f:

f.write(str(arr\_quick))

A2. Исходный код файла **Generator.py**

from faker import Faker

from Lab1 import FlightsArr

import random

faker = Faker()

Company = {"Компания": ["AiroFlot", "Pobeda", "TurkichAirlains", "NordVind", "S7", "JoPanAirlains", "AmogusAir"]}

def gen(n):

"""

Генерирует n записей вида id-Компания-Дата\_прилета-Время\_прилета-Кол\_во\_пассажиров

:param n: количество генерируемых записей

:type n: int

:return lst: список сгенерированных записей

:rtype lst: list

"""

lst = []

for \_ in range(n):

lst.append(str(str(random.randint(0, 1000)) + "--" + random.choice(Company["Компания"]) + "--" +

str(faker.date\_this\_month()) + "--" + str(faker.time()) + "--" + str(random.randint(1, 300))))

return lst

if \_\_name\_\_ == "\_\_main\_\_":

selections\_lst = [100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000]

for j in selections\_lst:

with open(f"db\_{j}.txt", "w") as db:

for i in gen(j):

db.write(i + "\n")