**Правительство Российской Федерации**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Федеральное государственное автономное образовательное учреждение

высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Кафедра «Компьютерная безопасность»

**ОТЧЕТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №2**

**По дисциплине**

|  |
| --- |
| Работу выполнил  Студент группы СКБ 192 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.О.Ташкинов  подпись, дата  Работу проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.А. Драчев  подпись, дата  Москва 2022 |

**«Методы программирования»**

**Постановка задачи**

**Лабораторная работа №2**

1) Реализовать прямой и бинарный поиск заданного элемента в массиве объектов по ключу в соответствии с вариантом (ключом является первое НЕ числовое поле объекта).

2) Входные данные для поиска обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).

3) Выполнить поиск 7-10 раз на массивах разных размерностей от 100 и более (но не менее 100000). Засечь (программно) время поиска для следующих способов: прямой поиск, бинарный поиск в заранее отсортированном массиве, сортировка массива (наиболее эффективным методом из работы 2) и бинарный поиск в нем. По полученным точкам построить графики зависимости времени поиска от размерности массива.

4) Записать входные данные в ассоциативный массив multimap<key, object> и сравнить время поиска по ключу в нем с временем поиска из п.3. Добавить данные по времени поиска в ассоциативном массиве в общее сравнение с остальными способами и построить график зависимости времени поиска от размерности массива.

5) Сделать отчет, содержащий титульный лист, код программы со спецификациями каждого метода и подробными комментариями, графики скоростей поиска и выводы.

Вариант 14:

Массив данных с расписанием поездов:

Расписание содержит номер поезда, дата отправления, тип поезда, (скорый, пассажирский, товарный), время отправления, время в пути (сравнение по полям – дата отправления, время отправления, номер поезда, время в пути (по убыванию))

Поиск по дате отправления

1. **Алгоритм решения задачи**

Данная лабораторная работа была реализована на языке программирования Python 3.9 Графики были построены с помощью Wolfram Mathematica.

В файле Lab1.py определен класс FlightsArr , его конструктор, перегрузки операторов и алгоритм быстрой сортировки quick\_sort(arr: list) -> list.

В файле Generator.py реализован генератор данных из класса FlightsArr gen(n)

В файле Lab2.py реализованы алгоритмы сортировки слиянием, бинарный поиск, линейный поиск в функциях:

Linear\_serch(arr: list, object) -> int – линейный поиск

Binary\_serch\_withoutsort(arr: list, object)-> int – бинарный поиск без сортировки

Binary\_serch\_withsort(arr: list, object)->int – бинарный поиск с сортировкой

**2. Выполнение задания**

2.1. quick\_sort(arr: list)

Функция быстрой сортировки, реализующая алгоритм быстрой сортировки.

2.2. Linear\_serch(arr: list, object)

Функция выполняет линейный поиск по массиву. Алгоритм проходит по массиву, возвращая первое вхождение искомого элемента.

2.3 Binary\_serch\_withoutsort(arr: list, object), Binary\_serch\_withsort(arr: list, object)

Функции выполняют рекурсивный бинарный поиск. Первая функция сортирует уже отсортированный массив, вторая функция сортирует, после этого выполняет поиск.

2.4. Класс FlightsArr

Данный класс содержит в себе информацию о расписании прибывающих авиарейсах. В класс реализован конструктор, в классе перегружены все операторы сравнения.

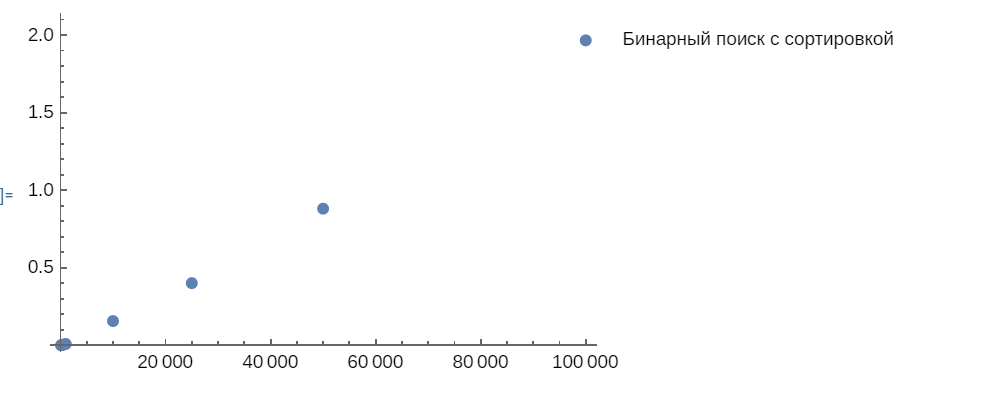
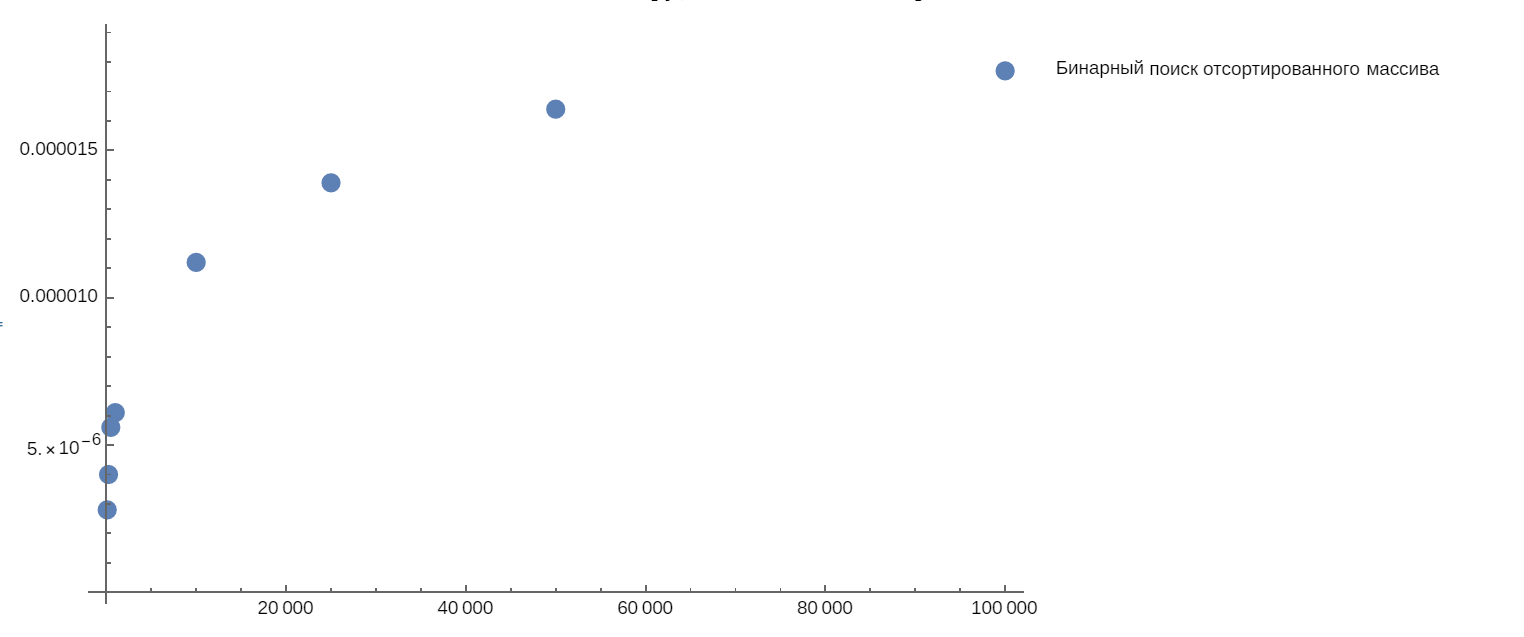
2.5. Функция gen (count: int)

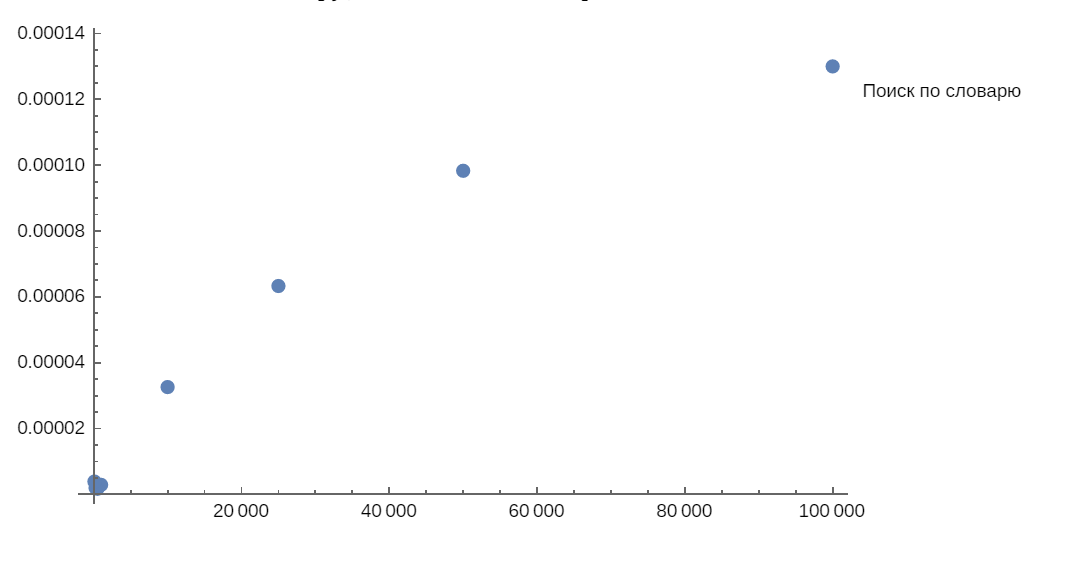
Генерирует случайный массив объектов класса FlightsArr. Принимает в качестве аргумента целое число, равное количеству объектов, которые нужно сгенерировать.

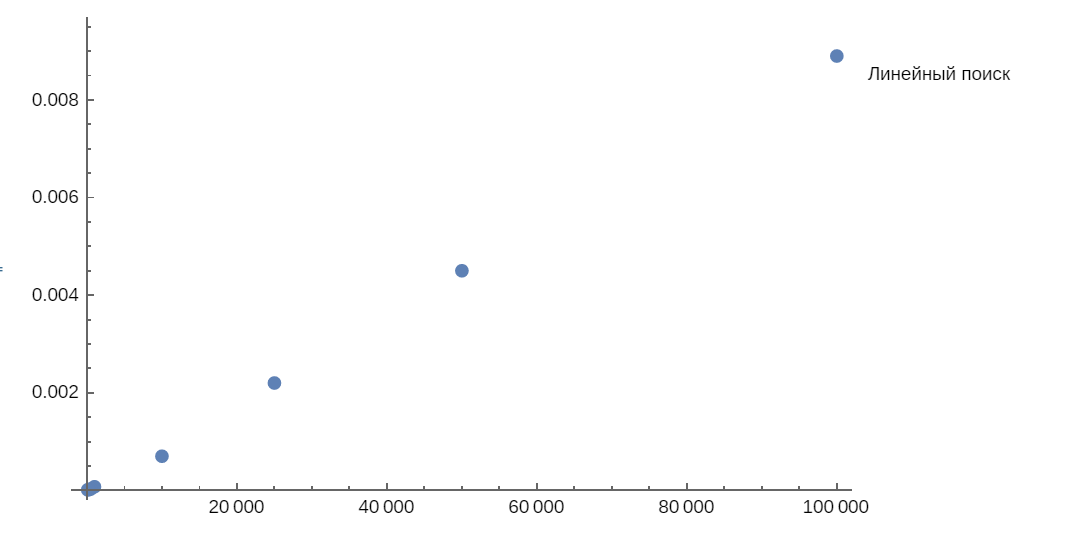
## Построение графиков и выводы

В результате работы программы были получены следующие данные

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Время линейного поиска, с | Время поиска по словарю, с | Время  бинарного поиска с сортировкой, с | Время бинарного поиска без сортировки, с |
| 100 | 1.1600\*10^(-5) | 3.9000\*10^(-6) | 0.0005 | 2.7999\*10^(-6) |
| 250 | 1.5899\*10^(-5) | 2.0000\*10^(-6) | 0.0018 | 4.0000\*10^(-6) |
| 500 | 2.7899\*10(-5) | 1.6999\*10^(-6) | 0.0045 | 5.6000\*10^(-6) |
| 1000 | 7.4299\*10(-5) | 2.9000\*10^(-6) | 0.0089 | 6.1000\*10^(-6) |
| 10000 | 0.0007 | 3.2600\*10^(-5) | 0.1566 | 1.1199\*10^(-5) |
| 25000 | 0.0022 | 6.3300\*10^(-5) | 0.4010 | 1.3900\*10^(-5) |
| 50000 | 0.0045 | 9.8299\*10^(-5) | 0.8809 | 1.6400\*10^(-5) |
| 1000000 | 0.0089 | 0.0001 | 1.9656 | 1.7700\*10^(-5) |







Временная теоретическая сложность алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время | Прямой поиск | Бинарный поиск с сортировкой | Бинарный поиск без сортировки | Поиск по словарю |
| Средний случай | O(n) | O(n\*logn) | O(logn) | O(logn) |

Как мы видим из таблицы, теоретическая сложность прямого поиска O(n), поиска по словарю O(logn), бинарного поиска с сортировкой O(n\*logn), бинарного поиска отсортированного массива O(logn), что согласовывается с полученными графиками. Так же можно сделать вывод, что если данные уже отсортированы, то лучше применять бинарный поиск, иначе прямой.

**Приложение А**

А.1 исходный код Lab2.py

from Lab1 import FlightsArr

from Lab1 import quick\_sort

import datetime

import time

import multidict

def Linear\_serch(arr: list, object) -> int:

"""

Линейный поиск

:param arr: массив объктов класса FlightsArr в котором происходит поиск

:type arr: list

:param object: ключ по которому ищем объект

:type object: string

:return: номер первого вхождения искомого обекта

:rtype: int

"""

for i in range(len(arr)): # проходим массив с первого элемента по последний

if(arr[i].name == object): # сравниваем эллемент массива с ключом

return i # возвращаем номер найденного элемента

return -1 # возвращаем -1 если элемент не найден

def Binary\_serch\_withoutsort(arr: list, object)-> int:

"""

Бинарный поиск у уже отсортированном массиве

:param arr: массив объктов класса FlightsArr в котором происходит поиск

:type arr: list

:param object: ключ по которому ищем объект

:type object: string

:return: номер первого вхождения искомого обекта

:rtype: int

"""

first = 0 # Индекс первого элемента

last = len(arr)-1 # Индекс последнего элемента

index = -1 # Номер искомого элемента

while (first <= last) and (index == -1): # Пока индекс поледнего элемента не равен индексу первого элемента или индекс искомого элемента равен не -1

mid = (first+last)//2 # Индекс среднего элемента

if arr[mid].name == object: # Если ключ равено среднему элементу

index = mid # Номер искомого элемента равен индексу среднегого

else: # Иначе

if object < arr[mid].name: # Если номер искомого элемента больше индекса среднегого

last = mid - 1 # Индекс последнего элмента становится равен индексу среднего элемента -1

else: # Иначе

first = mid + 1 # Индекс первого элмента становится равен индексу среднего элемента + 1

return index # Возвращаем индекс найденного элемента или -1 если такого нет

def Binary\_serch\_withsort(arr: list, object)->int:

"""

Бинарный поиск с быстрой сортировкой

:param arr: массив объктов класса FlightsArr в котором происходит поиск

:type arr: list

:param object: ключ по которому ищем объект

:type object: string

:return: номер первого вхождения искомого обекта

:rtype: int

"""

arr\_copy = quick\_sort(arr) # Сортируем исходный массив быстрой сортировкой

first = 0 # Индекс первого элемента

last = len(arr\_copy) - 1 # Индекс последнего элемента

index = -1 # Номер искомого элемента

while (first <= last) and (index == -1): # Пока индекс поледнего элемента не равен индексу первого элемента или индекс искомого элемента равен не -1

mid = (first + last) // 2 # Индекс среднего элемента

if arr\_copy[mid].name == object: # Если ключ равено среднему элементу

index = mid # Номер искомого элемента равен индексу среднегого

else: # Иначе

if object < arr\_copy[mid].name: # Если номер искомого элемента больше индекса среднегого

last = mid - 1 # Индекс последнего элмента становится равен индексу среднего элемента -1

else: # Иначе

first = mid + 1 # Индекс первого элмента становится равен индексу среднего элемента + 1

return index # Возвращаем индекс найденного элемента или -1 если такого нет

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}:")

t\_start\_1 = time.perf\_counter()

linear = Linear\_serch(arr.copy(), "Fake")

all\_time\_1 = time.perf\_counter() - t\_start\_1

print(f"Линейный поиск {all\_time\_1}")

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

t\_start\_2 = time.perf\_counter()

binary = Binary\_serch\_withoutsort(arr\_copy, "Fake")

all\_time\_2 = time.perf\_counter() - t\_start\_2

print(f"Бинарный поиск в сортированно массиве {all\_time\_2}")

t\_start\_3 = time.perf\_counter()

binaryws = Binary\_serch\_withsort(arr.copy(), "Fake")

all\_time\_3 = time.perf\_counter() - t\_start\_3

print(f"Бинарный писк с бытрой сортировкой {all\_time\_3}")

d = multidict.MultiDict()

for i in range(len(arr)):

d.add(str(arr[i].name), i)

t\_start\_4 = time.perf\_counter()

mul\_dic = d.get("fake", -1)

all\_time\_4 = time.perf\_counter() - t\_start\_4

print(f"Поиск по словарю {all\_time\_4}")

А.2 исходный код 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))

А.2 исходный код 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")