**Правительство Российской Федерации**

**Государственное образовательное бюджетное учреждение**

**высшего профессионального образования**

**«Научно-исследовательский университет –**

**Высшая школа экономики»**

**Факультет: МИЭМ**

**Направление: *Компьютерная безопасность***

**Отчет по лабораторной работе №2 (алгоритмы сортировки)**

**по дисциплине**

**«Методы программирования»**

**Выполнил**

**Студент группы СКБ151**

**Михалицын Пётр**

**Москва, 2017.**

Содержание

[Описание проекта 2](#_Toc500630168)

[Спецификация модулей 3](#_Toc500630169)

[Алгоритмы поиска и принципы работы с классом для осуществления поиска 6](#_Toc500630170)

[Результаты, полученные после оценки алгоритмов поиска и выводы 7](#_Toc500630171)

[Коды модулей 10](#_Toc500630172)

[Hash\_search\_db.py 10](#_Toc500630173)

[schedule\_maker.py 12](#_Toc500630174)

[search\_db.py 14](#_Toc500630175)

[sort\_db.py 15](#_Toc500630176)

[hasher.py 16](#_Toc500630177)

[sqlwrap.py 17](#_Toc500630178)

[train.py 19](#_Toc500630179)

[train\_schedule.py 21](#_Toc500630180)

[collision\_analysis.py 27](#_Toc500630181)

[hash\_test.py 29](#_Toc500630182)

[search\_test.py 31](#_Toc500630183)

[sort\_test.py 33](#_Toc500630184)

# Описание проекта

Проект заключался в сравнении трех видов поиска

1. Линейный
2. Бинарный
3. Поиск по ключу / с использованием хеш таблицы (стандартный)

Проект содержит в себе несколько модулей, которые можно разделить на 3 типа

1. Основные
   1. sort\_db.py
   2. hash\_search\_db.py
   3. search\_db.py
   4. schedulem\_maker.py
2. Вспомогательные
   1. sqlwrep.py
   2. train.py
   3. train\_schedule.py
   4. hasher.py
3. Тесты
   1. collision\_analysis.py
   2. hash\_test.py
   3. search\_test.py
   4. sort\_test.py

Все основные модули находятся в корневом каталоге проекта

Все вспомогательные модули находятся в директории source

Все тестовые модули находятся в каталоге tests

Вся информация сохраняется в базе данных SQLite

В ходе лабораторной работы нам будут интересны только 3 модуля: search\_db.py, train\_schedule.py, search\_test.pt; остальные подверглись незначительным изменениям после первого проекта. Посмотреть полностью весь проект можно в репозитории <https://github.com/lo1ol/train-schedule>

# Спецификация модулей

Для начала перечислим спецификацию всех новых функций и методов.

Модуль train\_schedule.py и обновленный класс Schedule

class Schedule:  
 *"""  
 Attribute database consist wrapping database with type DatabaseWrap* ***:param*** *database\_name: is str, consist path to database  
 Class consist  
 Attribute db: contain all data, before using call method load\_database  
 Attribute database: contain connection with database  
 Special Method \_\_init\_\_,  
 Special Method \_\_getattr\_\_ for raise exceptions if some functions are not called  
 Method load\_database for establish connection with database and load all records in list  
 Method unload\_database for sever connection with database and load all records in db to sqlite database  
 Method insert\_sort for insert sorting,  
 Method quick\_sort for quick sorting,  
 Method print\_schedule for printing formatted schedule  
 Static method \_format\_record for formatting record to string  
 Static method \_verify\_time\_format to verify time verify searching*

*Static method \_linear\_search for implementation linear search  
 Static method \_binary\_search for implementation binary search  
 Static method \_show\_result for show result in standard output stream  
 Method linear\_search for linear search in database  
 Method binary\_search for binary search in database  
 Method convert\_to\_dict for add opportunity of search by key  
 Method map\_search for search by key  
 Method convert\_to\_simple\_hash\_table for converting array to hash-table composed via simple hash  
 Method convert\_to\_rs\_hash\_table for converting array to hash-table composed via rs hash  
 Method simple\_hash\_search to search in hash-table via simple hash  
 Method rs\_hash\_search to search in hash-table via rs hash  
 """*

Статический метод \_linear\_search для реализации линейного поиска

def \_linear\_search(db, time):  
 *"""  
 Method for linear search* ***:param*** *time: for search by time in field time* ***:return****: list of results  
 """*

Метод linear\_search для линейного поиска в массиве записей расписаний поездов

def linear\_search(self, time, show=False):  
 *"""  
 Method for linear search* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """*

Метод binary\_search для линейного поиска в массиве записей расписаний поездов

def binary\_search(self, time, show=False):  
 *"""  
 Method for binary search* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """*

Статический метод \_binary\_search реализующий сам алгоритм бинарного поиска

def \_binary\_search(db, time):  
 *"""  
 Method for binary search* ***:param*** *time: for search by time in field time* ***:return****: list of results  
 """*

Метод convert\_to\_dict конвертирующий исходный массив в ассоциативный массив/словарь

def convert\_to\_dict(self):  
 *"""  
 Method convert\_to\_dict for add opportunity of search by key* ***:return****:   
 """*

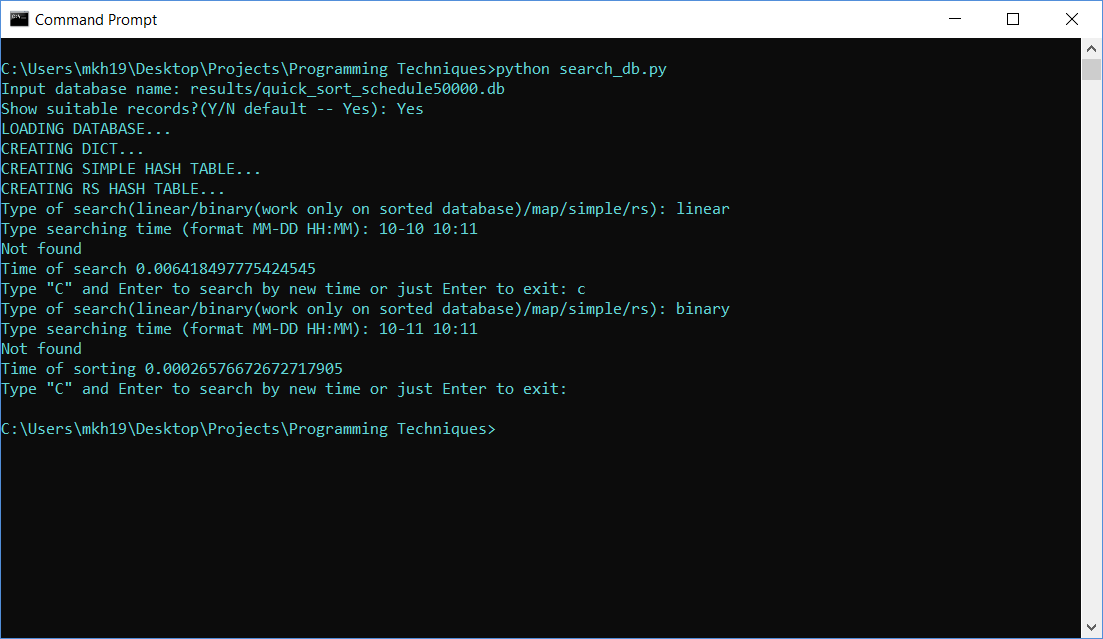
Метод map\_search реализует поиск в ассоциативном массиве

def map\_search(self, time, show=False):  
 *"""  
 Method for search by key* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """*

Модуль search\_db.py позволяет осуществлять поиск в базе данных расписаний с помощью командной строки выбирая в какой базе данных мы хотим искать, хотим ли выводить результаты поиска, какой метод поиска мы хотим использовать и делает при этом замер времени. Модуль содержит единственную функцию search\_db

def search\_db():  
 *"""  
 search record in database, which path input from stdin* ***:return****: 0   
 """*

Вот пример работы с данным модулем:



Последний модуль search\_test.py является тестовым и тестрирует на время различные алгоритмы поиска. Он содержит единственную функцию search. Результаты поиска записываются в лог-файл, который потом парсится программой, написанной в пакете mathematica. На основе лог-файла строятся все таблицы и графики

def search(log='logs/log\_search.txt', trace=False):  
 *"""  
 Make measure of two algorithms of search and make log file according this measure  
 Measure timing of computing 5, 10, 50, 100 and so on to 10\*\*5 of sorted records* ***:param*** *log: path to logfile* ***:param*** *trace: flag to showing measure in standard output stream* ***:return****: 0   
 """*

Сравнение производилось на базах данных разных размерностей. Причем все алгоритмы поиска применялись к одним и тем же базам данных и искались одни и те же записи, чтобы исключить все непредвиденные преимущества при поиске с помощь одного алгоритма над другим

# Алгоритмы поиска и принципы работы с классом для осуществления поиска

Прежде чем начать поиск следует выгрузить все данные из базы данных sqlite и создать на основе них массив. При этом для корректного поиска с помощью бинарного алгоритма следует загружать или уже отсортированную базу данных или сортировать всю базу данных перед поиском. При поиске с помощью ассоциативного массива сортировку производить не обязательно, но нужно создать ассоциативный массив перед началом поиска вызвав соответствующий метод, дабы не нарваться на исключение. Для линейного поиска никаких особенных ограничений не накладывается.

Линейный поиск ищет в нашем массиве элементы последовательно, причем даже если массив отсортирован, он продолжит поиск даже после нахождения всех записей удовлетворяющих поиску, что свидетельствует о том, что время поиска будет зависеть только от количества элементов в базе данных и не будет зависеть от местонахождения искомых записей в ней

Бинарный же поиск будет зависеть, как и от местонахождения элементов в базе данных и размера базы данных. Реализован он тривиально, как и обычный бинарный поиск.

Поиск в ассоциативном массиве реализован тривиально тоже, за основу ассоциативного массива брался объект типа defaultdict, чтобы не терять некоторые записи в случае коллизий

# Результаты, полученные после оценки алгоритмов поиска и выводы

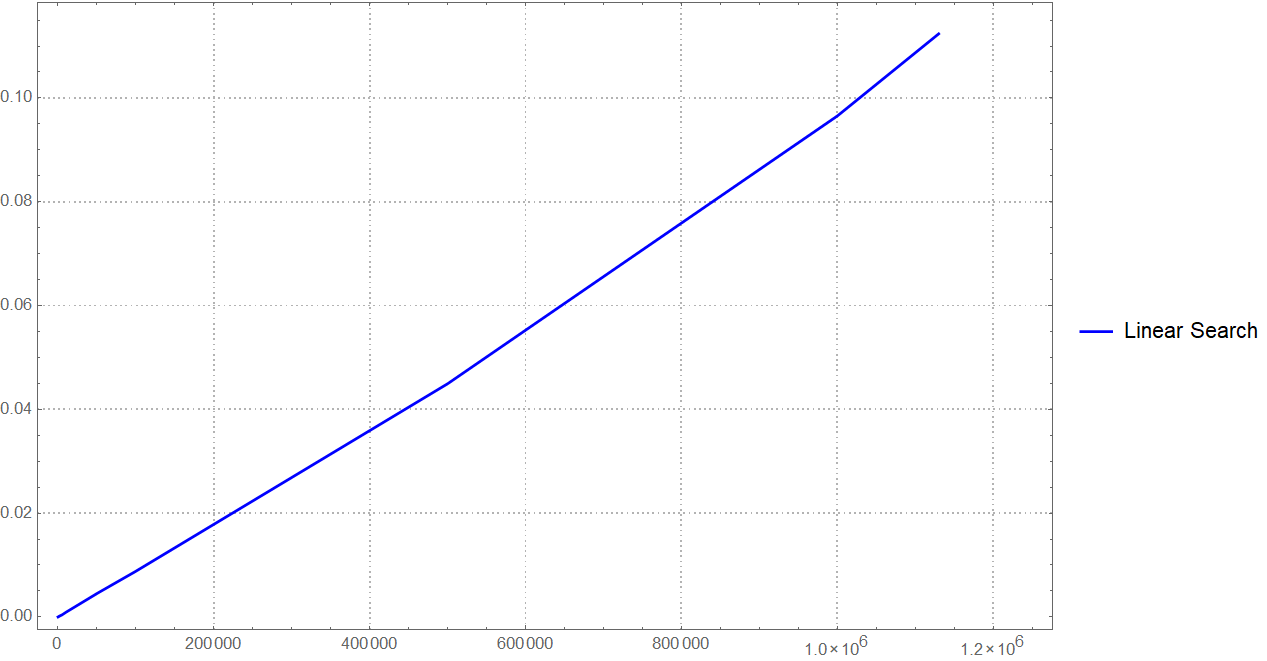
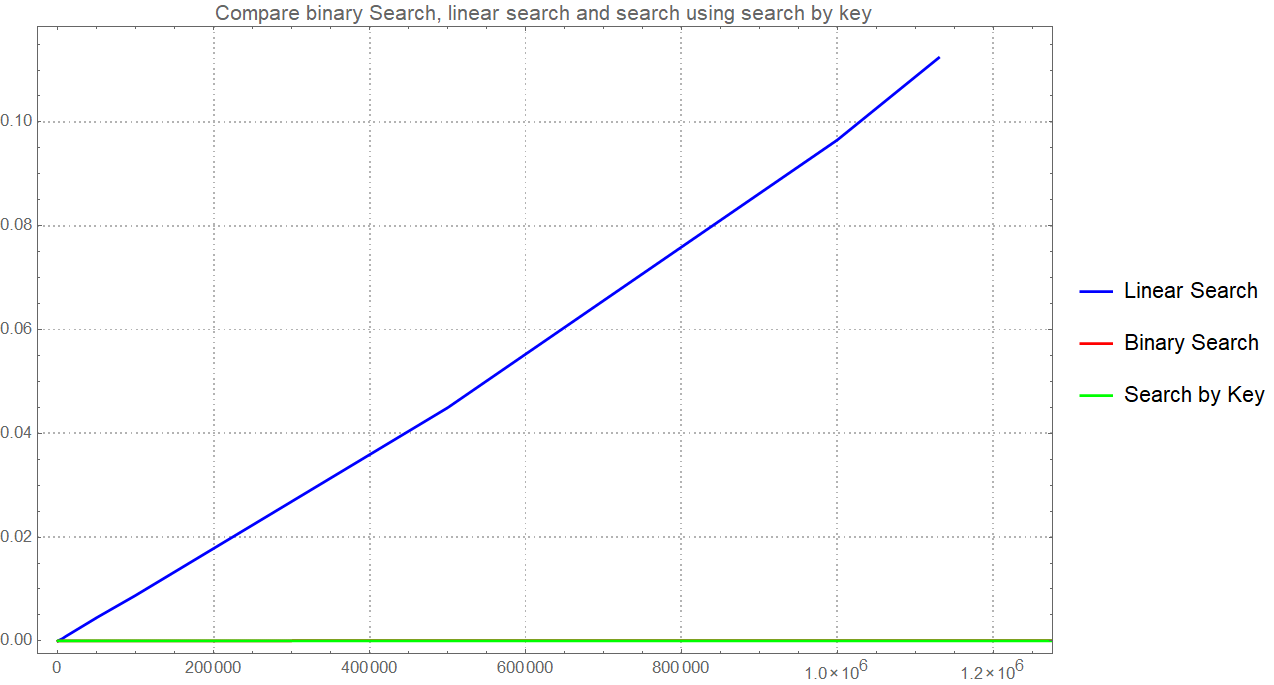
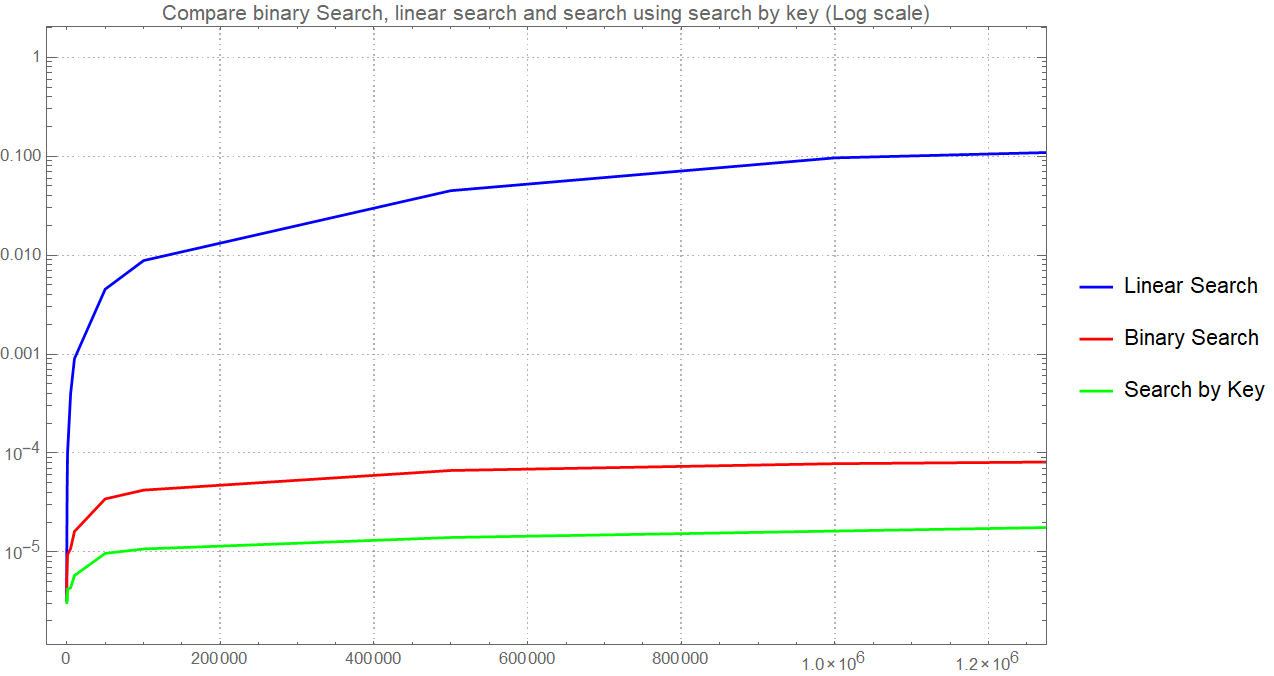
Как и следовало ожидать **линейный поиск имел линейную сложность O(N)**, связано как раз это было с тем, что все **записи просматривались последовательно**, более того немало сыграл тот факт, что поиск происходил в отсортированных массивах даже после того, когда искомые элементы были найдены.

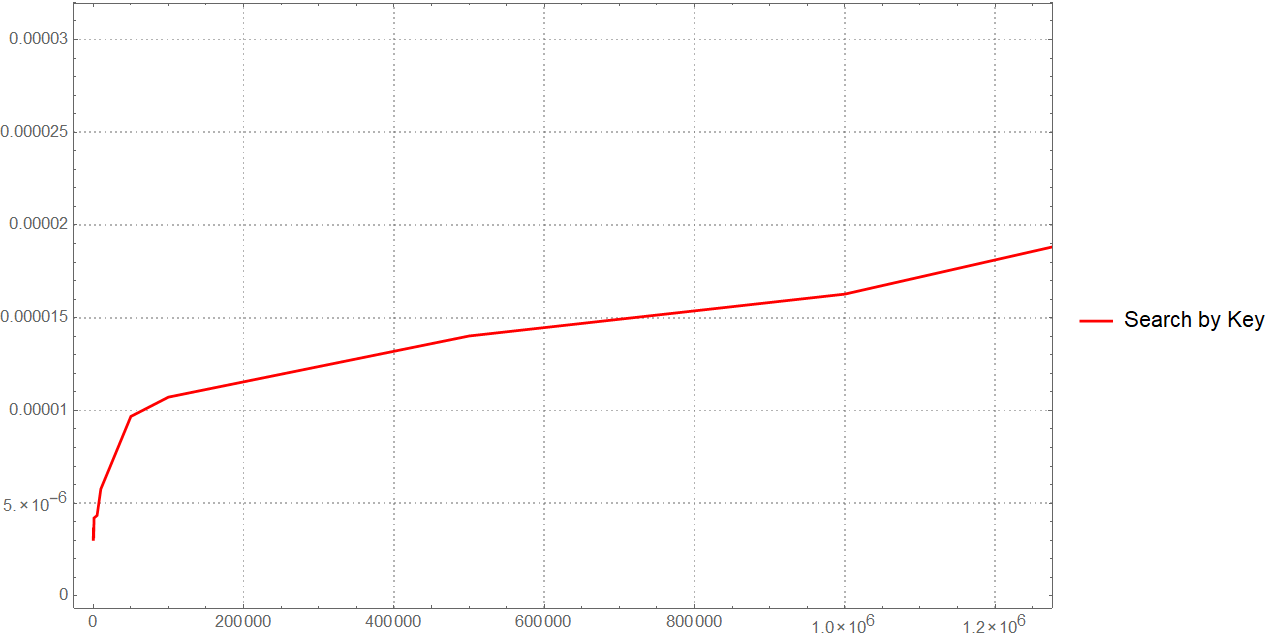
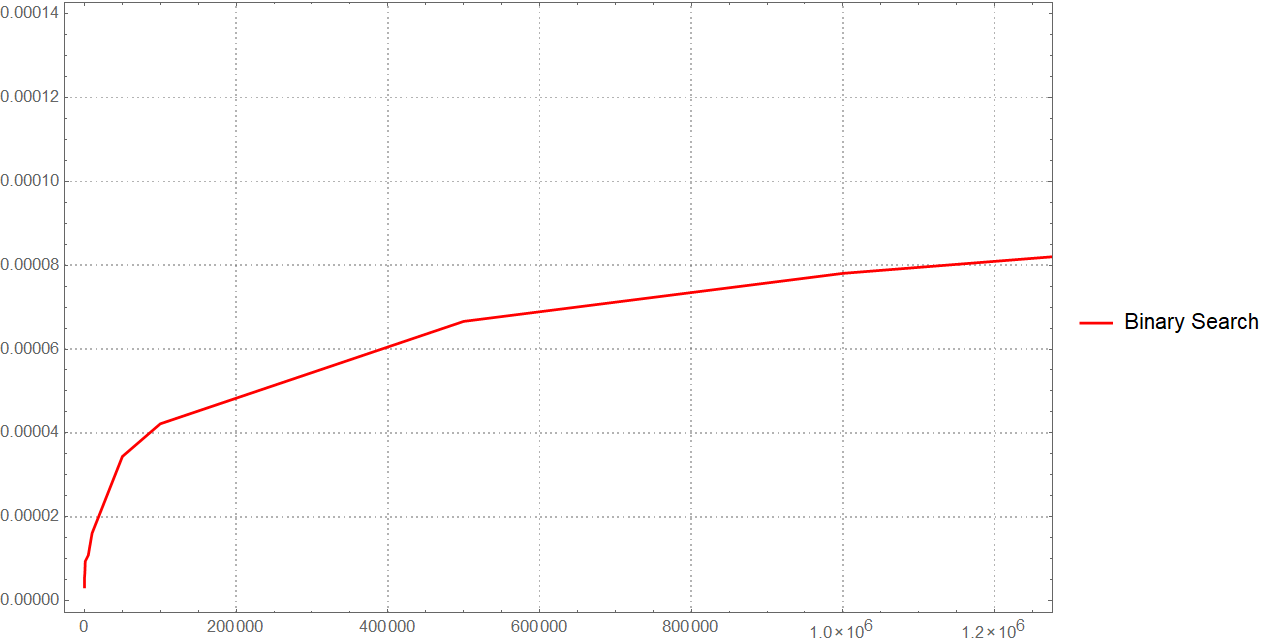
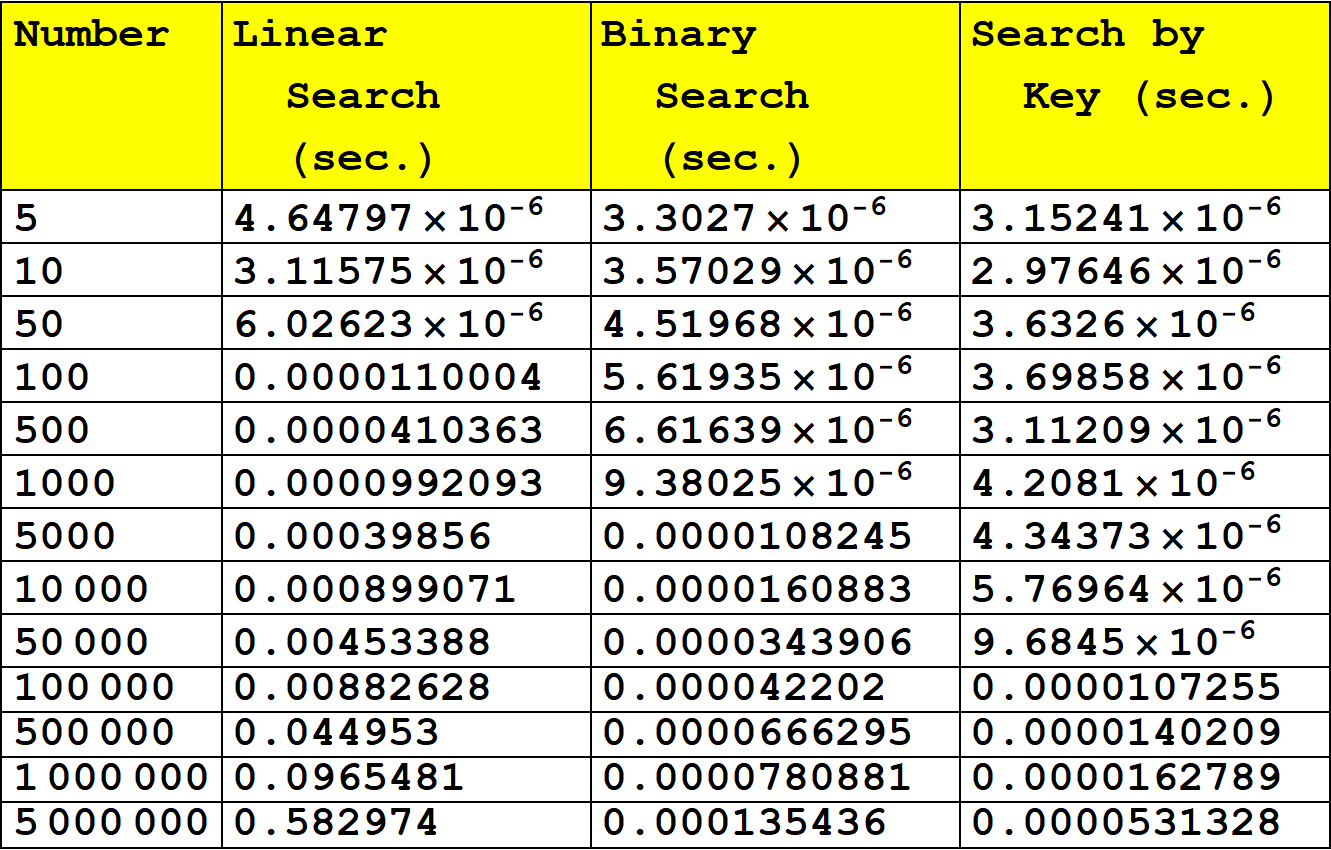
Бинарный поиск получил сложность **примерно логарифмическую сложность O(log(N)).** Связанно же это с тем, что **на каждой итерации массив делился пополам** и продолжался поиск тем же методом в нужной половинке массива.

**Поиск с помощью ассоциативного массива имел почти константную сложность O(1)** (хоть время и увеличивалось с увеличением n, но это увеличение было настолько незначительным, что им можно пренебречь, вероятно оно было связанно с устройством самого компьютера и алгоритма поиска нужных записей в оперативной памяти + образование коллизий). Связанно это с тем, что по сути время поиска зависело только от трех параметров – преобразование строки со временем в хеш, используя стандартный алгоритм в python (операция занимает константное время), обращение по индексу в массиве (константное время). И поиск внутри коллизий (скорее всего поиск внутри них происходит с помощью бинарного поиска, но это не факт и зависит от реализации). Поэтому по идее поиск в итоге должен иметь логарифмическую сложность, но т.к., как мы узнаем из следующей лабораторной работы, число коллизий очень мало, то поиск осуществляется почти полностью за константное время.

Стоит отметить, что поиск происходил в каждой базе данных 100 раз и по результатам бралось среднее выборочное и заносилось в лог-файл.

Снизу приведены графики и таблицы результатов поиска с использованием различных методов поиска





.

# Коды модулей

## Hash\_search\_db.py

from time import clock  
  
from source.train\_schedule import Schedule  
  
  
def hash\_search\_db():  
 *"""  
 search record in database, which path input from stdin* ***:return****: 0   
 """* database = input('Input database name: ')  
 show = input('Show suitable records?(Y/N default -- Yes): ')  
 if show.lower() == 'n':  
 show = False  
 elif show.lower() == 'y':  
 show = True  
 print('LOADING DATABASE...')  
 schedule = Schedule(database)  
 schedule.load\_database()  
 while True:  
 type = input('Type of search(simple/rs/default): ')  
 time = input('Type searching time (format MM-DD HH:MM): ')  
 if type == 'linear':  
 start = clock()  
 schedule.linear\_search(time, show=show)  
 print('Time of search %s' % (clock() - start))  
 elif type == 'binary':  
 start = clock()  
 schedule.binary\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 elif type == 'map':  
 schedule.convert\_to\_dict()  
 start = clock()  
 schedule.map\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 else:  
 print('Unknown Search!')  
 test = input('Type "C" and Enter to search by new time or just Enter to exit: ')  
 if test.lower() != 'c':  
 break  
 return 0  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 search\_db()

## schedule\_maker.py

import random  
import sqlite3  
  
  
def make\_database(name, db=None):  
 *"""  
 Make database on storage  
 Record is id, train number, type, depart time, travel time* ***:param****:name : int if want to get \*number\* random records  
 or str, that consist path to database file* ***:param****:db: None, if set name with type int. Further get value of name  
 db, if want to get database store records in db* ***:return****: -> name of created db  
   
 If name is number and db is None create number of random records and return ""./data/random\_schedule{name}.db  
 If name is string and db is number, create number of random records and return name  
 If name is string and db is list of records, return database with with records (with keeping order) and return name  
 """* if isinstance(name, int):  
 db = name  
 name = './data/random\_schedule%s.db' % db  
 if not db:  
 raise RuntimeError('Expected number if records')  
  
 connection = sqlite3.connect(name)  
 cursor = connection.cursor()  
 try:  
 cursor.execute('DROP TABLE schedule;')  
 except sqlite3.OperationalError:  
 pass  
  
 cmd\_make\_table = """  
 CREATE TABLE schedule(  
 id **INTEGER** PRIMARY KEY,  
 train\_number **INTEGER**,  
 type\_of\_train **VARCHAR**(9),  
 departure\_time **DATE**,  
 travel\_time **TIME**  
 )  
 """  
 cursor.execute(cmd\_make\_table)  
  
 cmd\_insert\_field = """  
 INSERT INTO schedule (id, train\_number, type\_of\_train, departure\_time, travel\_time)  
 VALUES (NULL, {train\_number},"{type}", "{d\_time}", "{t\_time}")"""  
  
 # Make database with records  
 if isinstance(db, list):  
 for train in db:  
 cursor.execute(cmd\_insert\_field.format(\*\*train.form()))  
 connection.commit()  
 connection.close()  
 return name  
  
 # Create random records and set in database  
 for i in range(db):  
 type = random.choice(['Express', 'Passenger'])  
 time = random.randint(0, 1439)  
 month = random.randint(1, 12)  
 day = random.randint(1,30)  
 d\_time = "%02d-%02d %02d:%02d" % (month, day, time // 60, time % 60)  
 if type == 'Express':  
 n = random.randint(360, 540)  
 t\_time = "%02d:%02d" % (n//60, n%60)  
 else:  
 n = random.randint(480, 720)  
 t\_time = "%02d:%02d" % (n//60, n%60)  
  
 cursor.execute(cmd\_insert\_field.format(train\_number=i, type=type, d\_time=d\_time, t\_time=t\_time))  
  
 connection.commit()  
 connection.close()  
 return name  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 name = input('Type name of database: ')  
 number = input("Type number of random records: ")  
 try:  
 make\_database(name, int(number))  
 except ValueError:  
 print("Not an integer number!")

## search\_db.py

from time import clock  
  
from source.train\_schedule import Schedule  
  
  
def search\_db():  
 *"""  
 search record in database, which path input from stdin* ***:return****: 0   
 """* database = input('Input database name: ')  
 show = input('Show suitable records?(Y/N default -- Yes): ')  
 if show.lower() == 'n':  
 show = False  
 elif show.lower() == 'y':  
 show = True  
 print('LOADING DATABASE...')  
 schedule = Schedule(database)  
 schedule.load\_database()  
 print('CREATING DICT...')  
 schedule.convert\_to\_dict()  
 print('CREATING SIMPLE HASH TABLE...')  
 schedule.convert\_to\_simple\_hash\_table()  
 print('CREATING RS HASH TABLE...')  
 schedule.convert\_to\_rs\_hash\_table()  
 while True:  
 type = input('Type of search(linear/binary(work only on sorted database)/map/simple/rs): ')  
 time = input('Type searching time (format MM-DD HH:MM): ')  
 if type == 'linear':  
 start = clock()  
 schedule.linear\_search(time, show=show)  
 print('Time of search %s' % (clock() - start))  
 elif type == 'binary':  
 start = clock()  
 schedule.binary\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 elif type == 'map':  
 start = clock()  
 schedule.map\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 elif type == 'simple':  
 start = clock()  
 schedule.simple\_hash\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 elif type == 'rs':  
 start = clock()  
 schedule.rs\_hash\_search(time, show=show)  
 print('Time of sorting %s' % (clock() - start))  
 else:  
 print('Unknown Search!')  
 test = input('Type "C" and Enter to search by new time or just Enter to exit: ')  
 if test.lower() != 'c':  
 break  
 return 0  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 search\_db()

## sort\_db.py

from time import clock  
  
from source.train\_schedule import Schedule  
  
  
def sort\_db():  
 *"""  
 Sort database, which path input from stdin* ***:return****: 0   
 """* database = input('Input batadase name: ')  
 type= input('Type of sort(quick/insert): ')  
 schedule = Schedule(database)  
 schedule.load\_database()  
 start = clock()  
 if type == 'quick':  
 schedule.quick\_sort()  
 print('Time of sorting %s' % (clock() - start))  
 elif type == 'insert':  
 schedule.insert\_sort()  
 print('Time of sorting %s' % (clock() - start))  
 else:  
 print('Unknown Sort!')  
 sorted\_data\_base = input('Input name for out file or type "No" to continue without saving: ')  
 if sorted\_data\_base != "No":  
 schedule.unload\_database(sorted\_data\_base)  
 return 0  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 sort\_db()

## hasher.py

def simple\_hash(value):  
 *"""  
 Simple hash function* ***:param*** *value: some string* ***:return****: hash of string from 0 to 4294967295  
 """* hash = 0  
 for num, char in enumerate(value):  
 hash += ord(char)\*\*(num+1) % 4294967296  
 return hash % 4294967296  
  
  
def rs(value):  
 *"""  
 RS hash function* ***:param*** *value: some string* ***:return****: hash of string from 0 to 4294967295  
 """* b, a, hash = 378551, 63689, 0  
 for i in value:  
 hash = (hash\*a+ord(i)) % 4294967296  
 a \*= b  
 return hash % 4294967296  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 from time import clock  
 start = clock()  
 hash1 = simple\_hash('12-30 23:59')  
 hash2 = simple\_hash('05-12 22:13')  
 print(clock()-start)  
 print(hash1)  
 print(hash2)  
 start = clock()  
 hash1 = rs('12-30 23:59')  
 hash2 = rs('05-12 22:13')  
 print(clock() - start)  
 print(hash1)  
 print(hash2)

## sqlwrap.py

import sqlite3  
  
"""  
Module contains class DatabaseWrap  
"""  
  
  
class DatabaseWrap:  
 *"""* ***:argument*** *database\_name:  
 Attribute database\_name  
 Attribute connection contain connect with database  
 Attribute cursor conatin cursor with database  
 2 Special-methods for index operator  
 Methods open, clods and getsize  
 """* def \_\_init\_\_(self, database\_name):  
 *"""* ***:param*** *database\_name: must contain database name if database doesn't exist make new  
 """* self.database\_name = database\_name  
  
 def \_\_getitem\_\_(self, item):  
 *"""  
 Provide access to database* ***:param*** *item: is id-1 of record, which return* ***:return****: tuple contain record with id == item+1 (without id)  
 """* cmd\_select = """SELECT \* FROM schedule WHERE id = %s"""  
 self.cursor.execute(cmd\_select % (item+1))  
 return self.cursor.fetchone()[1:]  
  
 def \_\_setitem\_\_(self, key, value):  
 *"""  
 Replace or make new record with id == key + 1 with values == value in database* ***:param*** *key: is id-1* ***:param*** *value: tuple (train\_number, type\_of\_train, departure\_time, travel\_time)* ***:return****: None  
 """* cmd\_replace = """REPLACE INTO schedule (id, train\_number, type\_of\_train, departure\_time, travel\_time)  
 VALUES ({0}, {1}, "{2}", "{3}","{4}") """  
 self.cursor.execute(cmd\_replace.format(key+1, \*value))  
  
 def \_\_len\_\_(self):  
 *"""* ***:return****: count of records in database  
 """* self.cursor.execute("SELECT *COUNT*(\*) FROM schedule")  
 return self.cursor.fetchone()[0]  
  
 def close(self):  
 *"""  
 Close connection with database and commit changes* ***:return****: None  
 """* self.connection.commit()  
 self.cursor.close()  
  
 def open(self):  
 *"""  
 Open connection with database and commit changes  
 and put in attribute connection connection with database  
 and in attribute cursor -- cursor* ***:return****: None  
 """* self.connection = sqlite3.connect(self.database\_name)  
 self.cursor = self.connection.cursor()

## train.py

from .hasher import\*  
  
class Train:  
 *"""  
 Consist values necessary for identification one train  
 instance of Train more then other if depart time is later  
 and if depart time is equal, train with more travel time is more  
 Also have method form for getting tuple, identified train (necessary for sqlite)   
 """* def \_\_init\_\_(self, number, type, d\_time, t\_time):  
 *"""  
 Make instance of Train* ***:param*** *number: train number* ***:param*** *type: Express or Passenger* ***:param*** *d\_time: department time* ***:param*** *t\_time: travel time  
 """* self.number = number  
 self.type = type  
 self.d\_time = d\_time  
 self.t\_time = t\_time  
 # hash of d\_time via simple\_hash  
 self.hash1 = simple\_hash(d\_time)  
 # hash of d\_time via rs  
 self.hash2 = rs(d\_time)  
  
 def \_\_eq\_\_(self, other):  
 if (self.d\_time == other.d\_time and self.t\_time == other.t\_time and self.type == other.type  
 and self.number == other.number):  
 return True  
 else:  
 return False  
  
 def \_\_ne\_\_(self, other):  
 return not self.\_\_eq\_\_(other)  
  
 def \_\_gt\_\_(self, other):  
 if self.d\_time > other.d\_time:  
 return True  
 elif self.d\_time == other.d\_time and self.t\_time > other.t\_time:  
 return True  
 else:  
 return False  
  
 def \_\_ge\_\_(self, other):  
 return self.\_\_gt\_\_(other) or self.\_\_eq\_\_(other)  
  
 def \_\_lt\_\_(self, other):  
 return not self.\_\_ge\_\_(other)  
  
 def \_\_le\_\_(self, other):  
 return not self.\_\_gt\_\_(other)  
  
 def form(self):  
 *"""* ***:return****: dictionary with keys train\_number, type, d\_time, t\_time  
 """* return {'train\_number': self.number, 'type': self.type, 'd\_time': self.d\_time, 't\_time': self.t\_time}  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = Train(100, 'Express', '05-12 22:13', '04:30')  
 print(x.hash1)  
 print(x.hash2)

## train\_schedule.py

import sys  
  
from source.train import Train  
  
import schedule\_maker  
from source.sqlwrap import DatabaseWrap  
from re import fullmatch  
from collections import defaultdict  
from source.hasher import \*  
  
"""  
Module consist implamentation class Schedule  
"""  
  
  
class Schedule:  
 *"""  
 Attribute database consist wrapping database with type DatabaseWrap* ***:param*** *database\_name: is str, consist path to database  
 Class consist  
 Attribute db: contain all data, before using call method load\_database  
 Attribute database: contain connection with database  
 Special Method \_\_init\_\_,  
 Special Method \_\_getattr\_\_ for raise exceptions if some functions are not called  
 Method load\_database for establish connection with database and load all records in list  
 Method unload\_database for sever connection with database and load all records in db to sqlite database  
 Method insert\_sort for insert sorting,  
 Method quick\_sort for quick sorting,  
 Method print\_schedule for printing formatted schedule  
 Static method \_format\_record for formatting record to string  
 Static method \_verify\_time\_format to verify time verify searching  
 Static method \_linear\_search for implementation linear search  
 Static method \_binary\_search for implementation binary search  
 Static method \_show\_result for show result in standard output stream  
 Method linear\_search for linear search in database  
 Method binary\_search for binary search in database  
 Method convert\_to\_dict for add opportunity of search by key  
 Method map\_search for search by key  
 Method convert\_to\_simple\_hash\_table for converting array to hash-table composed via simple hash  
 Method convert\_to\_rs\_hash\_table for converting array to hash-table composed via rs hash  
 Method simple\_hash\_search to search in hash-table via simple hash  
 Method rs\_hash\_search to search in hash-table via rs hash  
 """* def \_\_init\_\_(self, database\_name):  
 *"""* ***:param*** *database\_name: is str, consist path to database  
 """* self.database = DatabaseWrap(database\_name)  
  
 def \_\_getattr\_\_(self, item):  
 if item == 'db' and 'db' not in self.\_\_dict\_\_:  
 raise RuntimeError('load\_database must be called firstly')  
 if item == 'db\_dict' and 'db\_dict' not in self.\_\_dict\_\_:  
 raise RuntimeError('convert\_to\_dict must be called firstly')  
 if item == 'simple\_hash\_table' and 'simple\_hash\_table' not in self.\_\_dict\_\_:  
 raise RuntimeError('convert\_to\_simple\_hash\_table must be called firstly')  
 if item == 'rs\_hash\_table' and 'rs\_hash\_table' not in self.\_\_dict\_\_:  
 raise RuntimeError('convert\_to\_rs\_hash\_table must be called firstly')  
  
 def load\_database(self):  
 *"""  
 Establish connection with database and load all records in attribute db* ***:return****: None  
 """* self.database.open()  
 db = self.database  
 self.db = [Train(\*db[i]) for i in range(len(db))]  
  
 def unload\_database(self, name=None):  
 *"""  
 Sever connection with database and load all records from db to sqlite database* ***:param*** *name: name of database or None (if doesn't want to save changes)* ***:return****: None* ***:raise*** *raise RunrimeError if attribute db doesn't exist  
 """* if not name:  
 self.database.close()  
 return None  
 schedule\_maker.make\_database(name, self.db)  
 self.database.close()  
 self.database = DatabaseWrap(name)  
  
 def insert\_sort(self, autounload\_into=None):  
 *"""  
 Implementation of insert sort  
 sort attribute db via insert sor* ***:param*** *autounload\_into: if consist path to file name, automatically load db in database* ***:return*** *None* ***:raise*** *raise RunrimeError if attribute db doesn't exist  
 """* for i in range(len(self.db)-1):  
 train = self.db[i]  
 next\_train = self.db[i+1]  
 if train < next\_train:  
 j = i  
 while j != -1 and self.db[j+1] > self.db[j]:  
 temp = self.db[j+1]  
 self.db[j+1] = self.db[j]  
 self.db[j] = temp  
 j -= 1  
  
 if autounload\_into:  
 self.unload\_database(autounload\_into)  
  
 def quick\_sort(self, db=None, first=True, autounload\_into=None):  
 *"""  
 Implamentation og quick sort,  
 sort db via quick sort algorithm* ***:param*** *db: for recursion call* ***:param*** *first: for mark first call* ***:param*** *autounload\_into: if consist path to file name, automatically load db in database* ***:return****: None* ***:raise*** *raise RunrimeError if attribute sb doesn't exist  
 """* if db == []:  
 return []  
 if first:  
 db = self.db  
 # Sort all records on three groups  
 center = db[len(db)//2]  
 left = []  
 right = []  
 mid = []  
  
 for i in db:  
 if i > center: left.append(i)  
 elif i < center: right.append(i)  
 else: mid.append(i)  
  
 try:  
 db = self.quick\_sort(left, first=False)+mid+self.quick\_sort(right,first=False)  
 except RecursionError:  
 sys.setrecursionlimit(sys.getrecursionlimit()\*1.5)  
  
 if not first: return db  
 self.db = db  
  
 if autounload\_into:  
 self.unload\_database()  
  
 def print\_schedule(self):  
 *"""  
 Method for print in stdio format string of records* ***:return****: format string of records* ***:raise*** *raise RunrimeError if attribute sb doesn't exist  
 """* for i in self.db:  
 print(self.\_format\_record(i))  
  
 @staticmethod  
 def \_format\_record(record):  
 *"""  
 Method for getting formatted string from record* ***:return****: format string of records* ***:raise*** *raise RunrimeError if attribute sb doesn't exist  
 """* return '{type:<10} train № {train\_number:04} departs on {d\_time} and will be {t\_time[0]}{t\_time[1]} hours {t\_time[3]}{t\_time[4]} minutes in travel'.format(\*\*record.form())  
  
 @staticmethod  
 def \_verify\_time\_format(time):  
 *"""  
 Method raise exception while format of time is incorrect* ***:param*** *time: time for check* ***:raise*** *raise RunrimeError format of time is incorrect  
 """* if not fullmatch('\d{2}-\d{2} \d{2}:\d{2}', time):  
 raise RuntimeError('Incorrect Time Format!')  
  
 @staticmethod  
 def \_show\_result(result):  
 *"""  
 Method for printing result in stdio* ***:param*** *result: result of search  
 """* if not result:  
 print('Not found')  
 else:  
 for train in result:  
 print(train)  
 @staticmethod  
 def \_linear\_search(db, time):  
 *"""  
 Method for linear search* ***:param*** *time: for search by time in field time* ***:return****: list of results  
 """* suitable = []  
 for train in db:  
 if train.d\_time == time:  
 suitable.append(train)  
 return suitable  
  
 def linear\_search(self, time, show=False):  
 *"""  
 Method for linear search* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """* self.\_verify\_time\_format(time)  
 suitable = self.\_linear\_search(self.db, time)  
 suitable = list(map(Schedule.\_format\_record, suitable))  
 if show:  
 self.\_show\_result(suitable)  
 return suitable  
  
 @staticmethod  
 def \_binary\_search(db, time):  
 *"""  
 Method for binary search* ***:param*** *time: for search by time in field time* ***:return****: list of results  
 """* suitable = []  
 first = 0  
 last = len(db)-1  
 while first <= last:  
 mid = (first+last) >> 1  
 if db[mid].d\_time == time:  
 break  
 if db[mid].d\_time < time:  
 last = mid-1  
 else:  
 first = mid+1  
 else:  
 # if no one found  
 return suitable  
 # searching first suitable train  
 first = mid  
 while first >= 0 and db[first].d\_time == time:  
 first -= 1  
 first += 1  
 # create list of suitable train  
 while first < len(db) and db[first].d\_time == time:  
 suitable.append(db[first])  
 first += 1  
 return suitable  
  
 def binary\_search(self, time, show=False):  
 *"""  
 Method for binary search* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """* self.\_verify\_time\_format(time)  
 result = self.\_binary\_search(self.db, time)  
 result = list(map(Schedule.\_format\_record, result))  
 if show:  
 self.\_show\_result(result)  
 return result  
  
 def convert\_to\_dict(self):  
 *"""  
 Method convert\_to\_dict for add opportunity of search by key* ***:return****:  
 """* self.db\_dict = defaultdict(list)  
 for train in self.db:  
 self.db\_dict[train.d\_time].append(train)  
  
 def map\_search(self, time, show=False):  
 *"""  
 Method for search by key* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """* self.\_verify\_time\_format(time)  
  
 result = self.db\_dict[time]  
 suitable = list(map(self.\_format\_record, result))  
 if show:  
 self.\_show\_result(suitable)  
 return suitable  
  
 def convert\_to\_simple\_hash\_table(self):  
 *"""  
 Method convert array to hash table using simple hash function  
 """* self.hash\_size = len(self.db)  
 self.simple\_hash\_table = [[] for \_ in range(self.hash\_size)]  
 for train in self.db:  
 self.simple\_hash\_table[train.hash1 % self.hash\_size].append(train)  
  
 def convert\_to\_rs\_hash\_table(self):  
 *"""  
 Method convert array to hash table using rs hash function  
 """* self.hash\_size = len(self.db)  
 self.rs\_hash\_table = [[] for \_ in range(self.hash\_size)]  
 for train in self.db:  
 self.rs\_hash\_table[train.hash2 % self.hash\_size].append(train)  
  
 def simple\_hash\_search(self, time, show=False):  
 *"""  
 Method for search in to hash table by time hashing via simple hash* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """* self.\_verify\_time\_format(time)  
 guess = self.simple\_hash\_table[simple\_hash(time) % self.hash\_size]  
 result = self.\_linear\_search(guess, time)  
 result = list(map(self.\_format\_record, result))  
 if show:  
 self.\_show\_result(result)  
 return result  
  
 def rs\_hash\_search(self, time, show=False):  
 *"""  
 Method for search in to hash table by time hashing via rs hash* ***:param*** *time: for search by time in field time* ***:param*** *show: for print formatted results in stdio (if no one fits print 'Not found')* ***:return****: list of results  
 """* self.\_verify\_time\_format(time)  
 guess = self.rs\_hash\_table[rs(time) % self.hash\_size]  
 result = self.\_linear\_search(guess, time)  
 result = list(map(self.\_format\_record, result))  
 if show:  
 self.\_show\_result(result)  
 return result

## collision\_analysis.py

import optparse  
import sys  
import os  
os.chdir('../')  
from schedule\_maker import make\_database  
  
from source.train\_schedule import Schedule  
  
  
def search(log='logs/log\_collision.txt', trace=False):  
 *"""  
 Make measure of number collisions in by hash in three hash function  
 Measure timing of computing 5, 10, 50, 100 and so on to 10\*\*5 of sorted records* ***:param*** *log: path to logfile* ***:param*** *trace: flag to showing measure in standard output stream* ***:return****: 0   
 """* logfile = open(log, 'w')  
  
 for i in (int((10 \*\* (i // 2)) / 2) if i % 2 == 0 else (10 \*\* (i // 2)) for i in range(2, 15)):  
 db\_name = make\_database(i)  
 schedule = Schedule(db\_name)  
 schedule.load\_database()  
 sp\_hash\_number = rs\_hash\_number = dt\_hash\_number = 0  
 hash\_dict\_sp = {}  
 hash\_dict\_rs = {}  
 hash\_dict\_dt = {}  
 for train in schedule.db:  
 sp\_hash = train.hash1  
 rs\_hash = train.hash2  
 dt\_hash = train.d\_time.\_\_hash\_\_()  
 for hash, dict, n in ((sp\_hash, hash\_dict\_sp, 1), (rs\_hash, hash\_dict\_rs, 2), (dt\_hash, hash\_dict\_dt, 3)):  
 if hash in dict:  
 if train.d\_time not in dict[hash]:  
 if n == 1:  
 sp\_hash\_number += 1  
 elif n == 2:  
 rs\_hash\_number += 1  
 else:  
 dt\_hash\_number += 1  
 dict[hash].append(train.d\_time)  
 else:  
 dict[hash] = [train.d\_time]  
  
 print('Simple hash has on %-7s records = %-5s collisions' % (i, sp\_hash\_number), file=logfile)  
 print('RS hash has on %-7s records = %-5s collisions' % (i, rs\_hash\_number), file=logfile)  
 print('Default hash has on %-7s records = %-5s collisions' % (i, dt\_hash\_number), file=logfile)  
 if trace:  
 print('Simple hash has on %-7s records = %-5s collisions' % (i, sp\_hash\_number))  
 print('RS hash has on %-7s records = %-5s collisions' % (i, rs\_hash\_number))  
 print('Default hash has on %-7s records = %-5s collisions' % (i, dt\_hash\_number))  
 schedule.unload\_database()  
 print('-' \* 50, file=logfile)  
 if trace:  
 print('-' \* 50)  
 logfile.flush()  
  
 logfile.close()  
 return 0  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = optparse.OptionParser()  
 parser.add\_option('-t', '--trace', action='store\_true',  
 help='Trace the measure in standard output, default is False', default=False)  
 parser.add\_option('-l', '--log', type='string', help='Logging measure in to FIlE, default in logs/log\_hash.txt',  
 default='logs/log\_collision.txt')  
 (options, args) = parser.parse\_args(sys.argv)  
 search(\*\*options.\_\_dict\_\_)

## hash\_test.py

import optparse  
import sys  
import os  
os.chdir('../')  
from random import randint  
from time import clock  
from source.train\_schedule import Schedule  
  
  
def hash\_search(log='logs/log\_hash.txt', trace=False):  
 *"""  
 Make measure of three algorithms of search in hash table and make log file according this measure  
 Measure timing of computing 5, 10, 50, 100 and so on to 10\*\*5 of sorted records* ***:param*** *log: path to logfile* ***:param*** *trace: flag to showing measure in standard output stream* ***:return****: 0   
 """* logfile = open(log, 'w')  
  
 for i in (int((10 \*\* (i // 2)) / 2) if i % 2 == 0 else (10 \*\* (i // 2)) for i in range(2, 15)):  
 db\_name = 'results\quick\_sort\_schedule%s.db' % i  
 schedule = Schedule(db\_name)  
 schedule.load\_database()  
 simple\_hash\_time = rs\_hash\_time = map\_time = 0  
 schedule.convert\_to\_dict()  
 schedule.convert\_to\_simple\_hash\_table()  
 schedule.convert\_to\_rs\_hash\_table()  
 for \_ in range(100):  
 time = randint(0, 1439)  
 month = randint(1, 12)  
 day = randint(1, 30)  
 time = "%02d-%02d %02d:%02d" % (month, day, time // 60, time % 60)  
  
 start = clock()  
 schedule.simple\_hash\_search(time)  
 simple\_hash\_time += clock() - start  
  
 start = clock()  
 schedule.rs\_hash\_search(time)  
 rs\_hash\_time += clock() - start  
  
 start = clock()  
 schedule.map\_search(time)  
 map\_time += clock() - start  
 print('Simple search on %-7s records = %-22s sec' % (i, simple\_hash\_time / 100), file=logfile)  
 print('RS search on %-7s records = %-22s sec' % (i, rs\_hash\_time / 100), file=logfile)  
 print('Default search on %-7s records = %-22s sec' % (i, map\_time / 100), file=logfile)  
 if trace:  
 print('Simple search on %-7s records = %-22s sec' % (i, simple\_hash\_time / 100))  
 print('RS search on %-7s records = %-22s sec' % (i, rs\_hash\_time / 100))  
 print('Default search on %-7s records = %-22s sec' % (i, map\_time / 100))  
 schedule.unload\_database()  
 print('-' \* 50, file=logfile)  
 if trace:  
 print('-' \* 50)  
 logfile.flush()  
  
 logfile.close()  
 return 0  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = optparse.OptionParser()  
 parser.add\_option('-t', '--trace', action='store\_true',  
 help='Trace the measure in standard output, default is False', default=False)  
 parser.add\_option('-l', '--log', type='string', help='Logging measure in to FIlE, default in logs/log\_hash.txt',  
 default='logs/log\_hash.txt')  
 (options, args) = parser.parse\_args(sys.argv)  
 hash\_search(\*\*options.\_\_dict\_\_)

## search\_test.py

import optparse  
import sys  
import os  
os.chdir('../')  
from random import randint  
from time import clock  
  
from source.train\_schedule import Schedule  
  
  
def search(log='logs/log\_search.txt', trace=False):  
 *"""  
 Make measure of two algorithms of search and make log file according this measure  
 Measure timing of computing 5, 10, 50, 100 and so on to 10\*\*5 of sorted records* ***:param*** *log: path to logfile* ***:param*** *trace: flag to showing measure in standard output stream* ***:return****: 0   
 """* logfile = open(log, 'w')  
  
 for i in (int((10 \*\* (i // 2)) / 2) if i % 2 == 0 else (10 \*\* (i // 2)) for i in range(2, 15)):  
 db\_name = 'results\quick\_sort\_schedule%s.db' % i  
 schedule = Schedule(db\_name)  
 schedule.load\_database()  
 linear\_sum = binary\_sum = map\_sum = 0  
 schedule.convert\_to\_dict()  
 for \_ in range(100):  
 time = randint(0, 1439)  
 month = randint(1, 12)  
 day = randint(1, 30)  
 time = "%02d-%02d %02d:%02d" % (month, day, time // 60, time % 60)  
  
 start = clock()  
 schedule.linear\_search(time)  
 linear\_sum += clock() - start  
  
 start = clock()  
 schedule.binary\_search(time)  
 binary\_sum += clock() - start  
  
 start = clock()  
 schedule.map\_search(time)  
 map\_sum += clock() - start  
  
 print('Linear search on %-7s records = %-22s sec' % (i, linear\_sum / 100), file=logfile)  
 print('Binary search on %-7s records = %-22s sec' % (i, binary\_sum / 100), file=logfile)  
 print('Map search on %-7s records = %-22s sec' % (i, map\_sum / 100), file=logfile)  
 if trace:  
 print('Linear search on %-7s records = %-22s sec' % (i, linear\_sum / 100))  
 print('Binary search on %-7s records = %-22s sec' % (i, binary\_sum / 100))  
 print('Map search on %-7s records = %-22s sec' % (i, map\_sum / 100))  
 schedule.unload\_database()  
 print('-' \* 50, file=logfile)  
 if trace:  
 print('-' \* 50)  
 logfile.flush()  
  
 logfile.close()  
 return 0  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = optparse.OptionParser()  
 parser.add\_option('-t', '--trace', action='store\_true',  
 help='Trace the measure in standard output, default is False', default=False)  
 parser.add\_option('-l', '--log', type='string', help='Logging measure in to FIlE, default in logs/log\_search.txt',  
 default='logs/log\_search.txt')  
 (options, args) = parser.parse\_args(sys.argv)  
 search(\*\*options.\_\_dict\_\_)

## sort\_test.py

import optparse  
import sys  
from time import clock  
import os  
os.chdir('../')  
from schedule\_maker import make\_database  
from source.train\_schedule import Schedule  
  
  
def sort(log='logs/log\_sort.txt', trace=False):  
 *"""  
 Make measure of two algorithms of sort and make log file according this measure  
 Before making measure ask user about confidence in action  
 Measure timing of computing 5, 10, 50, 100 and so on to 10\*\*5 of records for insert sort and 10\*\*7 for quick sort  
 For each number generate random schedule in sqlite* ***:param*** *log: path to logfile* ***:param*** *trace: flag to showing measure in standard output stream* ***:return****: 0   
 """* if input('Are you sure? All previous database and log will be removed(Type "Yes" if sure):') == 'Yes':  
 logfile = open(log, 'w')  
  
 for i in (int((10\*\*(i//2))/2) if i % 2 == 0 else (10\*\*(i//2)) for i in range(2, 15)):  
 db\_name = make\_database(i)  
 # For stop insert sort  
 if i <= 10\*\*5:  
 schedule = Schedule(db\_name)  
 schedule.load\_database()  
 start = clock()  
 schedule.insert\_sort()  
 print('Insert sort on %-7s records = %-22s sec' % (i, clock() - start), file=logfile)  
 if trace:  
 print('Insert sort on %-7s records = %-22s sec' % (i, clock() - start))  
 schedule.unload\_database('results/insert\_sort\_schedule%s.db' % i)  
 else:  
 print('Insert sort on %-7s records = %-22s sec' % (i, 'more then 3 hours'), file=logfile)  
 if trace:  
 print('Insert sort on %-7s records = %-22s sec' % (i, 'more then 3 hours'))  
 logfile.flush()  
 schedule = Schedule(db\_name)  
 schedule.load\_database()  
 start = clock()  
 schedule.quick\_sort()  
 print('Quick sort on %-7s records = %-22s sec' % (i, clock() - start), file=logfile)  
 if trace:  
 print('Quick sort on %-7s records = %-22s sec' % (i, clock() - start))  
 schedule.unload\_database('results/quick\_sort\_schedule%s.db' % i)  
 print('-'\*50, file=logfile)  
 if trace:  
 print('-' \* 60)  
 logfile.flush()  
 logfile.close()  
 return 0  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = optparse.OptionParser()  
 parser.add\_option('-t', '--trace', action='store\_true', help='Trace the measure in standard output, default is False',default =False)  
 parser.add\_option('-l', '--log', type='string', help='Logging measure in to FIlE, default in log\_sort.txt', default='logs/log\_sort.txt')  
 (options, args) = parser.parse\_args(sys.argv)  
 sort(\*\*options.\_\_dict\_\_)