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

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
    a. sort_db.py
    b. hash_search_db.py
    c. search_db.py
    d. schedulem_maker.py
2) Вспомогательные
    a. sqlwrep.py
    b. train.py
    c. train_schedule.py
    d. hasher.py
3) Тесты
    a. collision_analysis.py
    b. hash_test.py
    c. search_test.py
    d. 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:
    """
```

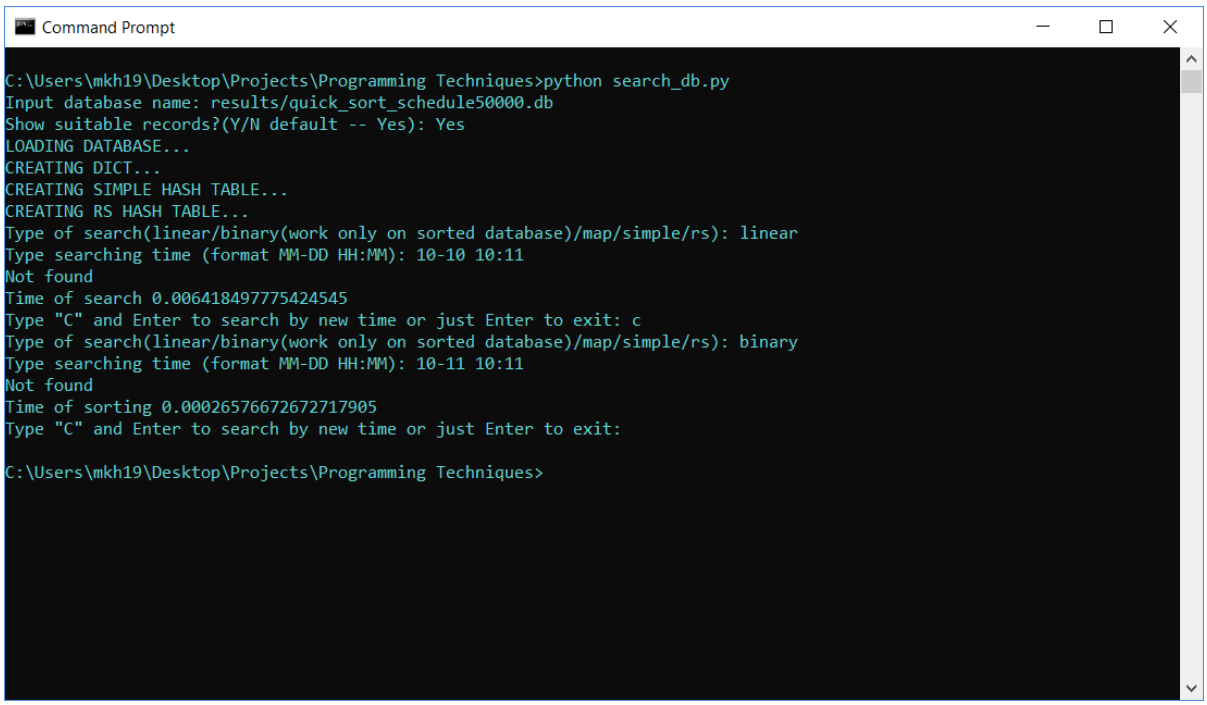
Метод map_search реализует поиск в ассоциативном массиве

```
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.

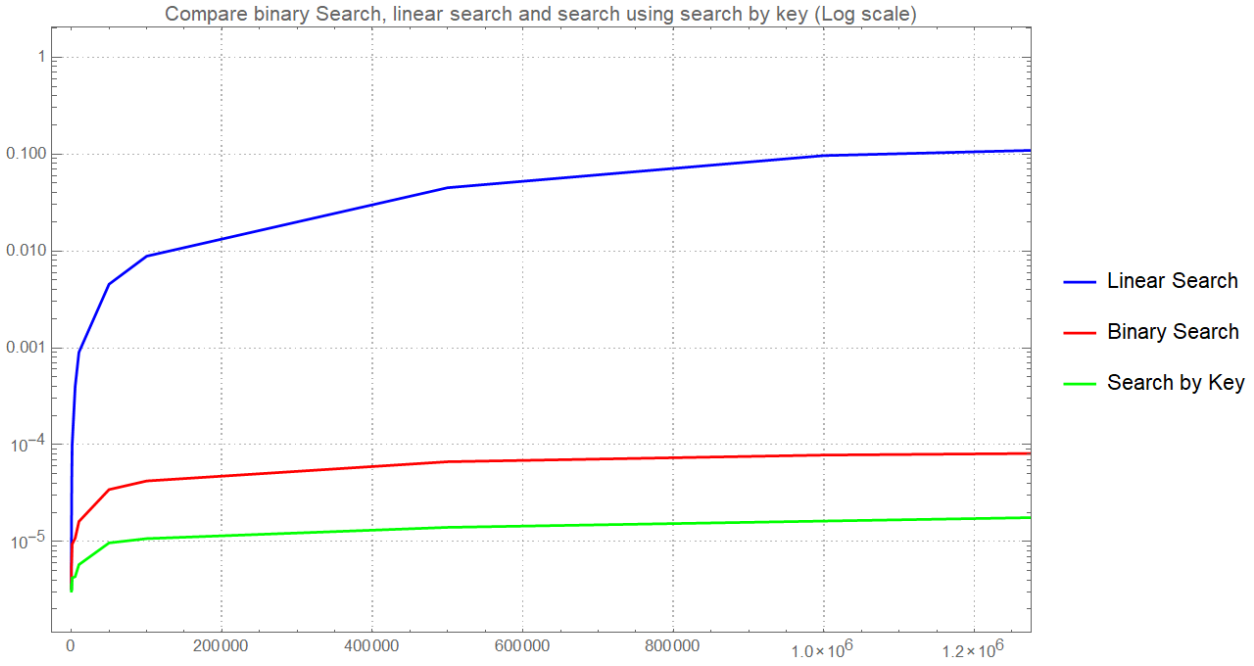
Бинарный поиск получил сложность **примерно логарифмическую сложность O(log(N)).** Связанно же это с тем, что **на каждой итерации массив делился пополам** и продолжался поиск тем же методом в нужной половинке массива.

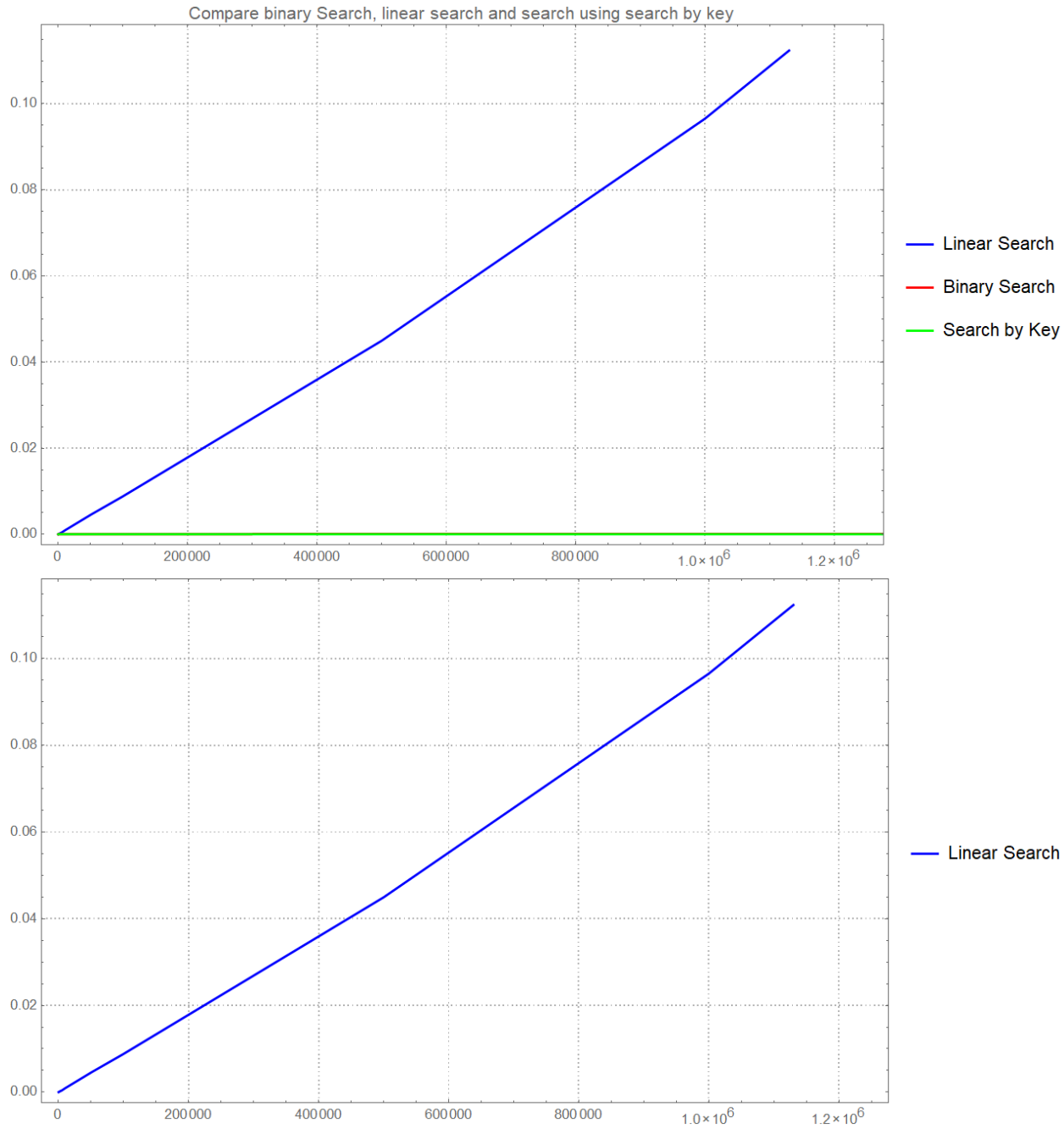
Поиск с помощью ассоциативного массива имел почти константную сложность O(1) (хоть время и увеличивалось с увеличением n, но это увеличение было настолько незначительным, что им можно пренебречь, вероятно оно было связанно с устройством самого компьютера и алгоритма поиска нужных записей в оперативной 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.

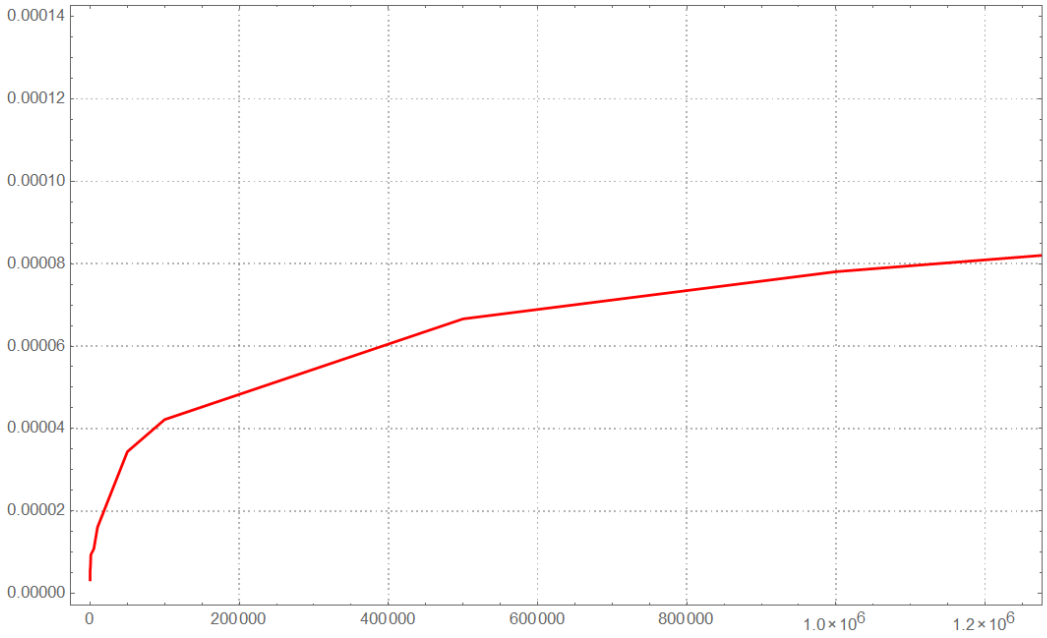
Снизу приведены графики и таблицы результатов поиска с использованием различных методов поиска

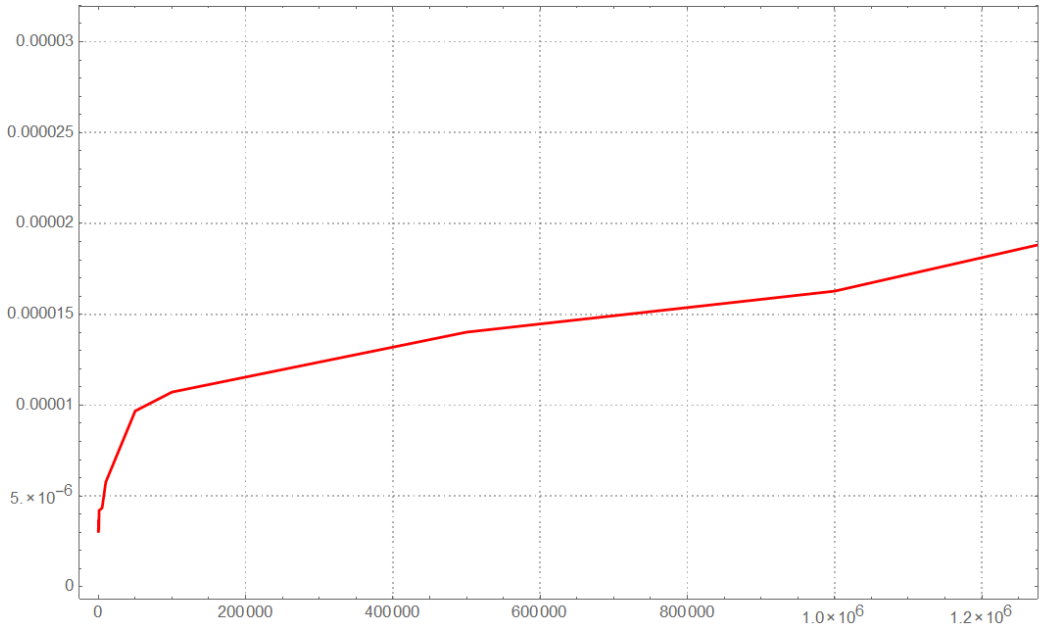
Number	Linear Search (sec.)	Binary Search (sec.)	Search by Key (sec.)
5	4.64797×10^{-6}	3.3027×10^{-6}	3.15241×10^{-6}
10	3.11575×10^{-6}	3.57029×10^{-6}	2.97646×10^{-6}
50	6.02623×10^{-6}	4.51968×10^{-6}	3.6326×10^{-6}
100	0.0000110004	5.61935×10^{-6}	3.69858×10^{-6}
500	0.0000410363	6.61639×10^{-6}	3.11209×10^{-6}
1000	0.0000992093	9.38025×10^{-6}	4.2081×10^{-6}
5000	0.00039856	0.0000108245	4.34373×10^{-6}
10 000	0.000899071	0.0000160883	5.76964×10^{-6}
50 000	0.00453388	0.0000343906	9.6845×10^{-6}
100 000	0.00882628	0.000042202	0.0000107255
500 000	0.044953	0.0000666295	0.0000140209
1 000 000	0.0965481	0.0000780881	0.0000162789
5 000 000	0.582974	0.000135436	0.0000531328




Compare binary Search, linear search and search using search by key
0.10
0.08
0.06
0.04
0.02
0.00
0
200000
400000
600000
800000
1.0×10^6
1.2×10^6
Linear Search
Binary Search
Search by Key

0.10
0.08
0.06
0.04
0.02
0.00
0
200000
400000
600000
800000
1.0×10^6
1.2×10^6
Linear Search





.

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__)
```