**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**](#_Toc495865886)

[**Спецификация модулей 3**](#_Toc495865887)

[**Взаимодействие модулей 8**](#_Toc495865888)

[**Результаты, полученные после оценки алгоритмов сортировки и выводы 9**](#_Toc495865889)

[**Пример базы данных до/после сортировки 11**](#_Toc495865890)

[**Коды модулей 12**](#_Toc495865891)

[main.py 12](#_Toc495865892)

[sort\_db.py 14](#_Toc495865893)

[schedule\_maker.py 15](#_Toc495865894)

[sqlwrap.py 17](#_Toc495865895)

[train.py 19](#_Toc495865896)

[train\_schedule.py 20](#_Toc495865897)

# Описание проекта

Проект заключался в сравнении двух видов сортировки

1. Вставками
2. Быстрая

Проект содержит в себе несколько модулей, которые можно разделить на 2 типа

1. Основные
   1. sort\_db.py
   2. schedule\_maker.py
   3. main.py
2. Вспомогательные
   1. sqlwrep.py
   2. train.py
   3. train\_schedule.py

Все вспомогательные модули находятся в директории source

Вся информация сохраняется в базе данных SQLite

# Спецификация модулей

Для начала перечислим спецификацию всех функций, классов и их методов, предоставляемых каждым модулем

**Модуль train.py** содержит класс Train, который обладает следующей спецификацией

**class Train**:  
 *"""  
 Consist values necessary for identification one train  
 instance of Train more than 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)   
 """*

Конструктор Train

**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  
 """*

Класс предоставляет перегрузки всех операторов, необходимые для сравнения двух объектов этого типа, по критерию указного в спецификации

Кроме того, класс предоставляет метод form, который возвращает словарь значения, необходимый для заполнения записи в записи в базе денных.

**def form**(self):  
 *"""* ***:return****: dictionary with keys train\_number, type, d\_time, t\_time  
 """*

**Модуль train\_schedule.py** – своего рода контейнер предоставляющий интерфейс для работы с базой данных.Модуль содержит реализацию класса Schedule со следующей спецификацией.

**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\_\_,  
 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-generator get\_schedule for getting format string of records  
 """*

Конструктором

**def** \_\_init\_\_(self, database\_name):  
 *"""* ***:param*** *database\_name: is str, consist path to database  
 """*

Методом load\_database необходимым для загрузки базы данных в атрибут массив db, находящегося в классе

**def load\_database**(self):  
 *"""  
 Establish connection with database and load all records in attribute db* ***:return****: None  
 """*

Методом unload\_database выгружающий атрибут массив db в базу данных, находящуюся на диске

**def unload\_database**(self, name):  
 *"""  
 Sever connection with database and load all records from db to SQLite database* ***:param*** *name: name of database* ***:return****: None*

***:raise*** *raise RunrimeError if attribute db doesn't exist*

*"""*

Методом insert\_sort, сортирующий атрибут массив db с помощью сортировки вставками

**def insert\_sort**(self, autounload\_into=**None**):  
 *"""  
 Implementation of insert sort  
 sort attribute db via insert sort* ***: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  
 """*

Методом quick\_sort, сортирующий атрибут массив db с помощью быстрой сортировки

**def quick\_sort**(self, db=**None**, first=**True**, autounload\_into=**None**):  
 *"""  
 Implementation of 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 db doesn't exist  
 """*

Методом-генератором get\_schedule, возвращающим форматированные записи в базе данных db

**def get\_schedule**(self):  
 *"""  
 Method-generator for getting format string of records* ***:return****: format string of records* ***:raise*** *raise RunrimeError if attribute db doesn't exist  
 """*

**Модуль sqlwrap.py** необходим для работы с базой данных обращение к которой инкапсулировано с помощью методов и перегруженных операторов класса DatabaseWrap, находящегося в этом модуле

**class DatabaseWrap**:  
 *"""* ***:argument*** *database\_name:  
 Attribute database\_name  
 Attribute connection contain connect with database  
 Attribute cursor contains cursor with database  
 2 Special-methods for index operator  
 Methods open, clods and getsize  
 """*

Конструктор

**def** \_\_init\_\_(self, database\_name):  
 *"""* ***:param*** *database\_name: must contain database name if database doesn't exist make new  
 """*

Перегрузка операций обращения по индексу для чтения обращения и обращения по индексу для записи

**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)  
 """*

**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  
 """*

Метод get\_size, предоставляющий возможность эффективно узнать кол-во записей в базе данных

**def get\_size**(self):  
 *"""* ***:return****: count of records in database  
 """*

Методы open и close, для начала и прекращения работы с базой данных

**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*

**def close**(self):  
 *"""  
 Close connection with database and commit changes* ***:return****: None  
 """*

**Модуль schedule\_maker.py** предоставляет функцию, необходимую для генерации случайной базы данных. Также в случае запуска модуля, как самостоятельной программы, имеется возможность создать базу данных интерактивно. Спецификация функции make\_database, находящейся в этом модуле

**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  
 """*

**Модуль main.py** является ключевым для расчета времени необходимого для сортировки и занесения результатов в лог. файл и при указании соответствующего флага вывода результатов в консоль (при запуске, как самостоятельной программы)

**def main**(log='log.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   
 """*

**Модуль sort\_db.py** а основном может использоваться как самостоятельная программа, которая предоставляет функцию sort\_db

**def sort\_db**():  
 *"""  
 Sort database, which path input from stdin* ***:return****: 0   
 """*

# Взаимодействие модулей

Модуль main.py сначала создает лог файл, в который будут записываться результаты времени сортировки. Перед каждой итерацией создается случайная база данных размерностью 5, 10, 50, 100, …, 10^6, 5\*10^6 х с помощью функции make\_database находящейся в модуле schedule\_maker.py. на каждой итерации создается путь к базе данных передается в конструктор класса Schedule. Каждый объект этого класса сначала загружает базу данных с диска, потом сортирует его с помощью insert\_sort, потом другая сущность того же типа инициализированной той же базой данных сортируется с помощью quick\_sort. Результаты заносятся в лог файл и при указании флага выводятся в поток вывода. На quick\_sort проверяются все базы данных. На Insert sort проверяются только базы данных размер, которых не больше 10^5 записей

Для модуля sort\_db.py реализован похожим образом, за исключением того, что ничего не выводится в лог файл, а выводится только в поток вывода и итерация у нас всего лишь одна и способ сортировки мы выбираем сами.

Объект класса Schedule проживает следующую жизнь (в основном. В нашем случае только такую). Сначала мы его инициализируем, потом загружаем в него базу данных (точнее в его атрибут db) с помощью соответствующего метода, потом db сортируется. И выгружается в базу данных.

db хранит в себе объекты класса Train которые можно сравнивать между собой. В самой же базе данных sqlite хранятся картежи необходимые для инициализации объектов класса Train. Класс DatabaseWrap в модуле sqlwrap.py предоставляет все необходимые методы для извлечения этих картежей (путем перегрузки операции индексирования).

Во время выгрузки же с помощью метода form класса Train получаются обратно картеж значений необходимых для инициализации и с помощью перегрузки операции присвоения по индексу заносятся в базу данных эти картежи.

# Результаты, полученные после оценки алгоритмов сортировки и выводы

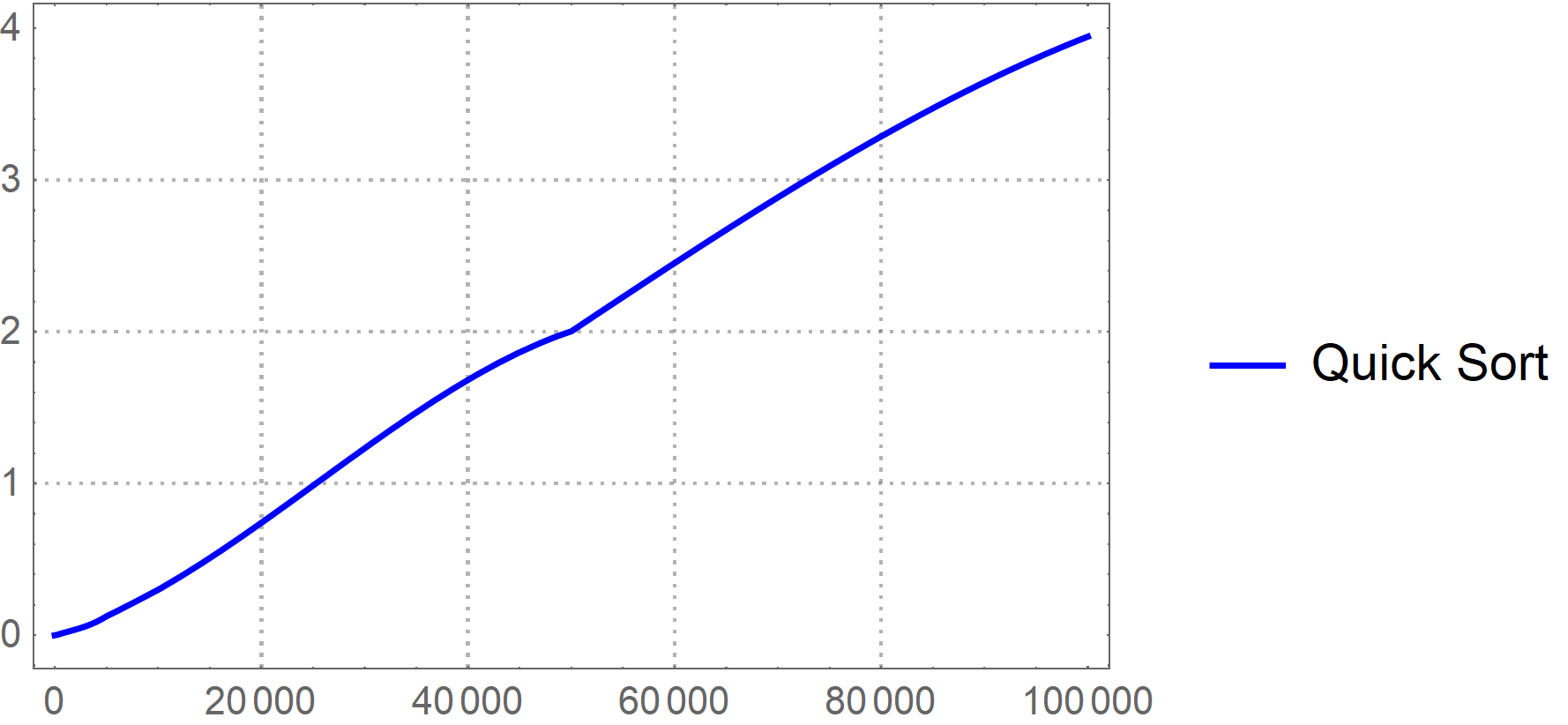
Как ожидалось **алгоритм вставками имеет алгоритмическую сложность O(n^2)**, но в то же время **является стойким** (из-за реализации) алгоритмом и при **обладающим естественностью**, т.к. при запуске на случайной базе данных размером 10^5 записей сортировал ее 3560 секунд. При сортировке же уже отсортированной таблицы сортировал ее 0.13 секунд. К тому же он **расходовал O(1) дополнительной памяти**, и поэтому является эффективны по памяти алгоритмом.

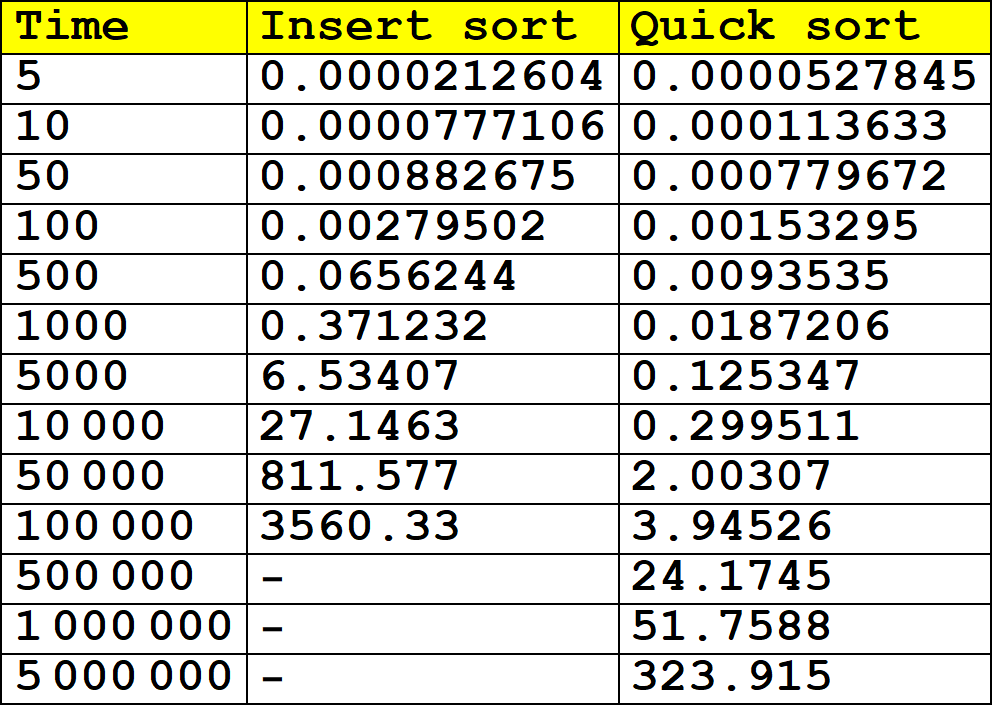
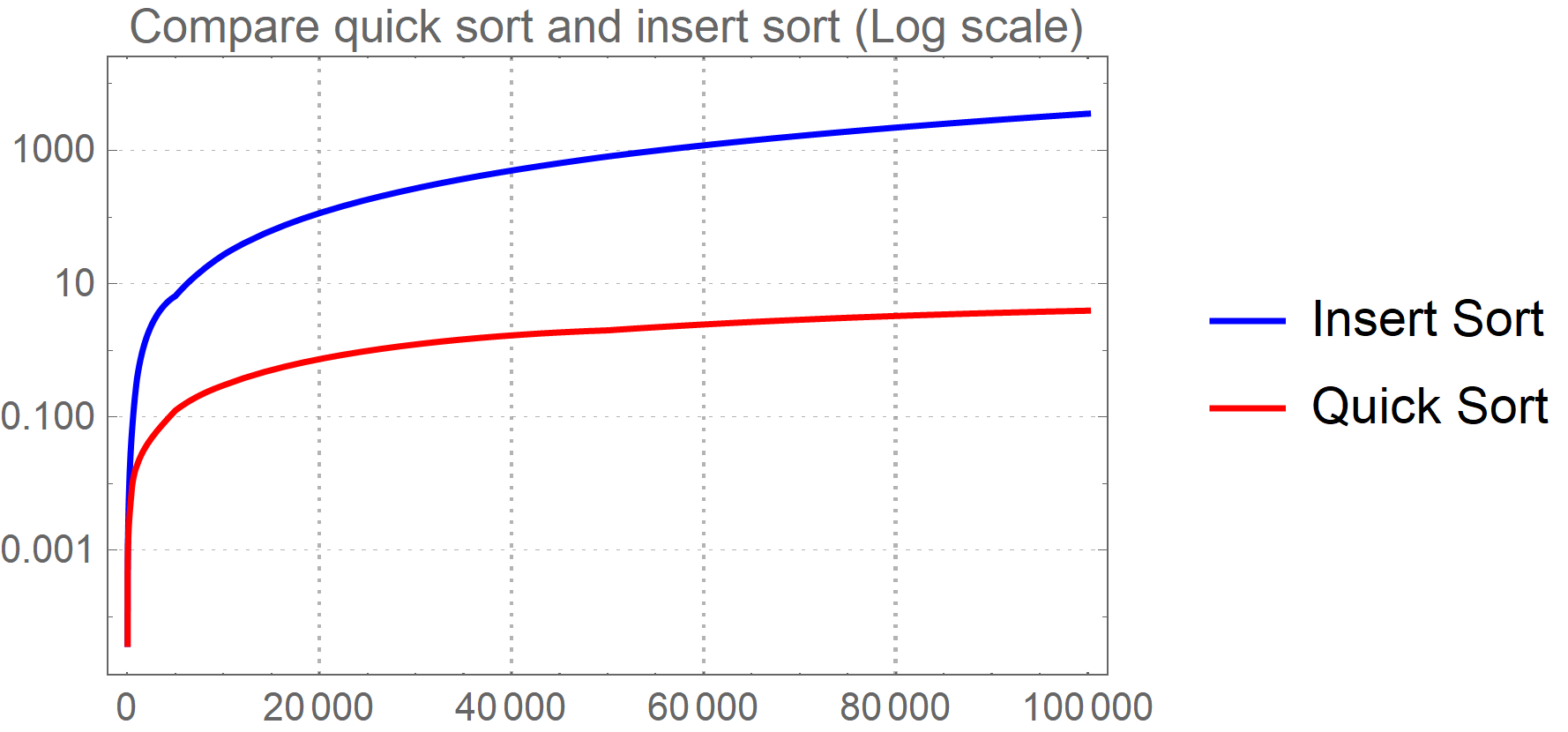
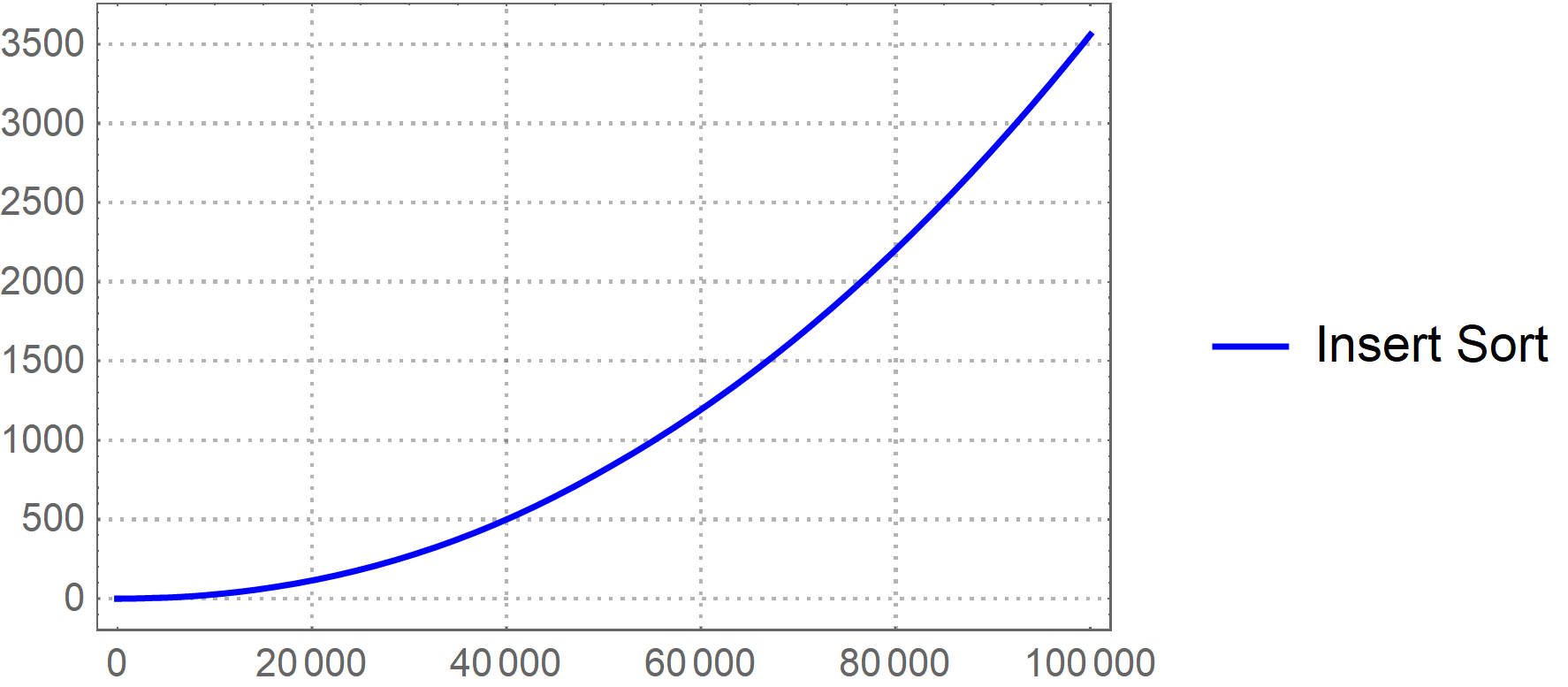
**Алгоритм же быстрой сортировки имеет сложность O(n\*log(n))** по времени. Но зато **не обладает стойкостью**, о чем свидетельствует данная запись в уже отсортированной базе данных (в неотсортированной базе данных, все номера поездов идут по порядку)



Второе поле – номер поезда. 4ое и 5ое время отправления и время в пути, соответственно. Таким образом три эквивалентных элемента не сохранили порядок номеров, хотя и являются эквивалентными. Так же этот алгоритм **расходует дополнительно O(log(n)) памяти** необходимой для “держания в голове” подмассивов записей. Данный алгоритм **является естественным**, но не столь естественным, как сортировка вставками. Естественность алгоритма выводится из того, что в подмассивах не надо производить никаких действий касаемых перемещения из одного подмассива в другой (больших и меньших опорного элемента). Так, например, для 10^5 элементов сортировка произошла в 2 раза быстрее на уже отсортированном массиве (2 секунды вместо 4). Но **в любом из случаев, естественно, что гораздо лучше использовать сортировку вставками т.к. для почти отсортированного массива она эффективна она эффективна по времени, памяти, стойкости.** В другую сторону **быструю сортировку можно было бы модернизировать, введя флаг, который показывал было ли на какой-то глубине вложенности произведено перекидывание элементов из одного подмасива в другой и остановить сортировку в случае лжи (перекидываний не было=> массив отсортировано)**. Таким образом мы время примерно равное тому же, что и получили бы при сортировке вставками.

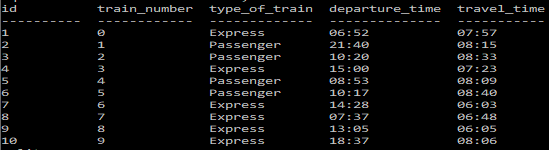
Ниже показаны графики и таблица сравнения эффективности этих алгоритмов по времени



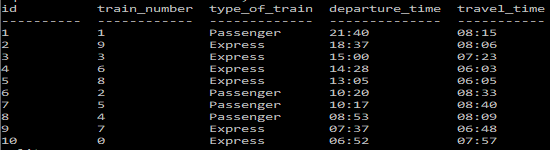


# Пример базы данных до/после сортировки

До:



После:



# Коды модулей

## main.py

**import** optparse  
**import** sys  
**from** time **import** clock  
  
**from** schedule\_maker **import** make\_database  
**from** source.train\_schedule **import** Schedule  
  
  
**def main**(log='log.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()  
 i = int(i\*3.74)  
  
 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.txt', default='log.txt')  
 (options, args) = parser.parse\_args(sys.argv)  
 main(\*\*options.\_\_dict\_\_)

## 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()

## 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 TIME,  
 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'])  
 n = random.randint(0, 1439)  
 d\_time = "%02d:%02d" % (n//60, n%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!")

## 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 get\_size**(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

**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  
  
 **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}

## train\_schedule.py

**import** sys  
  
**from** source.train **import** Train  
  
**import** schedule\_maker  
**from** source.sqlwrap **import** DatabaseWrap  
  
"""  
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\_\_,  
 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-generator get\_schedule for getting format string of records  
 """* **def** \_\_init\_\_(self, database\_name):  
 *"""* ***:param*** *database\_name: is str, consist path to database  
 """* self.database = DatabaseWrap(database\_name)  
  
 **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(db.get\_size())]  
  
 **def unload\_database**(self, name):  
 *"""  
 Sever connection with database and load all records from db to sqlite database* ***:param*** *name: name of database* ***:return****: None* ***:raise*** *raise RunrimeError if attribute db doesn't exist  
 """* **try**:  
 self.db  
 **except** NameError:  
 **raise** RuntimeError('load\_database must be called firstly')  
  
 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  
 """* **try**:  
 self.db  
 **except** NameError:  
 **raise** RuntimeError('load\_database must be called firstly')  
  
 **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  
 """* **try**:  
 self.db  
 **except** NameError:  
 **raise** RuntimeError('load\_database must be called firstly')  
  
 **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 get\_schedule**(self):  
 *"""  
 Method-generator for getting format string of records* ***:return****: format string of records* ***:raise*** *raise RunrimeError if attribute sb doesn't exist  
 """* **try**:  
 self.db  
 **except** NameError:  
 **raise** RuntimeError('load\_database must be called firstly')  
  
 **for** i **in** self.db:  
 **yield** '{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(\*\*i.form())