HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER ENGINEERING

BBM204
PROGRAMMING LABORATORY

Experiment 2

REPORT

Naciye Güzel 21580841

8 April 2017

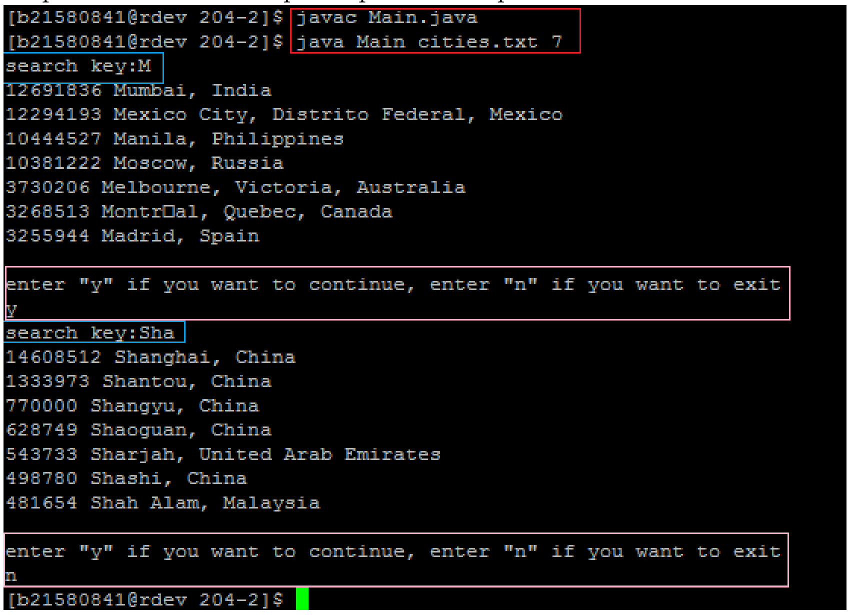
Contents

Chapter 1

Software Using Documentation

1.1 Software Usage

The software runs from the console. The input file and k should be given as arguments from the console respectively. After the program is run, the search key is given as standart input by user. Then the program repeatedly asks for new search key. The output is the results of the top k matching terms in descending order of weight. They are printed to the standart output. There are examples input and output as follows:



```
[b21580841@rdev 204-2]$ javac Main.java
[b21580841@rdev 204-2]$ java Main cities.txt 7
search key:M
12691836 Mumbai, India
12294193 Mexico City, Distrito Federal, Mexico
10444527 Manila, Philippines
10381222 Moscow, Russia
3730206 Melbourne, Victoria, Australia
3268513 Montr□al, Quebec, Canada
3255944 Madrid, Spain

enter "y" if you want to continue, enter "n" if you want to exit
y
search key:Sha
14608512 Shanghai, China
1333973 Shantou, China
770000 Shangyu, China
628749 Shaoguan, China
543733 Sharjah, United Arab Emirates
498780 Shashi, China
481654 Shah Alam, Malaysia

enter "y" if you want to continue, enter "n" if you want to exit
n
[b21580841@rdev 204-2]$
```

1.1.1 Compile-Run

javac Main.java
java Main input.txt k

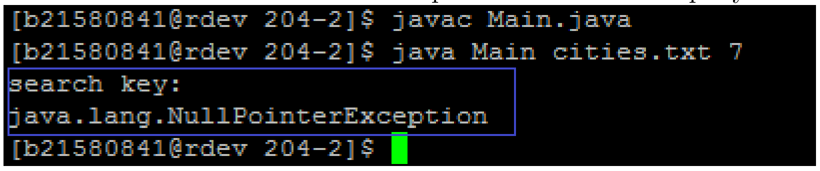
1.2 Provided Possibilities

This software can use easily everyplace. All code are not main method. It is written with methods from different classes. It is written to be clear in this way and understandable. It can develop easily by developer. Code writing follows the Java standards. Also, it is CASE-SENSITIVE.

1.3 Error Messages

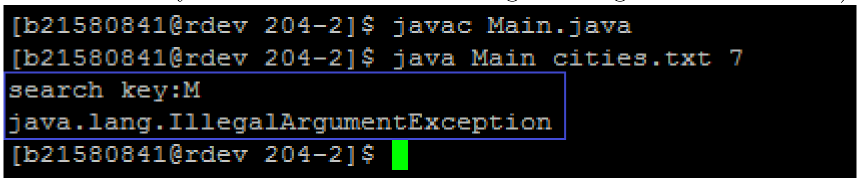
There are error messages. It does not accept everything that has been entered correctly. If there is an error, the program is terminated.

First one is for Null Pointer Exception. It is thrown if query is null as follows.



```
[b21580841@rdev 204-2]$ javac Main.java
[b21580841@rdev 204-2]$ java Main cities.txt 7
search key:
java.lang.NullPointerException
[b21580841@rdev 204-2]$
```

Second one is for Illegal Argument Exception. It is thrown if weight is negative. (Weight value of the city name that start M is changed to negative to see result.)



```
[b21580841@rdev 204-2]$ javac Main.java
[b21580841@rdev 204-2]$ java Main cities.txt 7
search key:M
java.lang.IllegalArgumentException
[b21580841@rdev 204-2]$
```

Chapter 2

Software Design Notes

2.1 Description of the program

2.1.1 Problem

The problem is to develop a system to implement the Complete Matching Words via Search Key. There are two arguments. The first one input file. This one is read by software. This file includes an integer N followed by N pairs of query strings and non negative weights. The weights and the query string are divided by tab. The second one is k that is number of wanted query to print in the descending order. Then, there is an standart input that is the wanted query of user to search from input file.
We were wanted a algorithm that complete Matching Words via Search Key according to given arguments that is input file and k, and standart input. It is mandatory to use Binary search algorithm in the search of the given word. But there is a condition which is to print the result of query in descending order of weight value according to k number.
At the end, the result of the query shall be printed to the standart output. Also, The program shall continue in repetitively according to user want.

2.1.2 Solution

My solution is that firstly, the given input file is read and it is splitten according to tab. For given the number of lines (I accept line object as query string(name) and weight in on line.), line objects are created. Then, the splitten lines are saved into object values that are weight and name of the line. To achieve binary seacrh, the objects have to be sorted. For this reason, the objects are sorted with Quick Sort according to their query name. Quicksort is used since it is guarantee fastest in practice. The reason to be fastest in practice is that Quicksort does not make unnecessary element swaps that are time consuming. So it has got less swap, more speed than the other sorting algorithms.
After sorting the objects, the query that is got from standart input is searched from the sorted objects with Binary Search algorithm. The found objects with binary search algorithm are sorted again with quick sort algorithm according to their weights. After that, the result(s) of query are printed on screen in descending order according to k argument.
All of the above are done for each repeated query. After each query is finished, the user is asked whether he/she wants to do a new query or not. So completing matching words are found easily

2.1.3 Algorithm

```
1. Get search key value
....Check whether key is empty
....If the key is empty, throw exception
```

```
2. Read the given file.
3. Get line number.
4. Create line objects array according to line number.
5. For each line object.
....Split the read line according to split
....Assign the weight and query name to object values
6. Sort the line objects according to query names.
7. Search the given key from the sorted objects
....If there is not any result, warn the user
....Check whether the found weight is negative
....If it is negative, throw exception
8. Sort the line objects according to weights.
9. Print out in descending order according to k value.
....Check whether found search number is less than k
....If it is less than k, print out according to number of found search.
```

2.1.4 Time Complexity

The Binary search has lgN time complexity. It uses compareTo() operations on keys. It is better than sequential search since it has got N/2 time complexity.
Not: The array to search have to sorted.
The Quicksort has time complexities as follows.

In worst case:

$$N^2/2 \tag{2.1}$$

In average case:

$$2NlnN \tag{2.2}$$

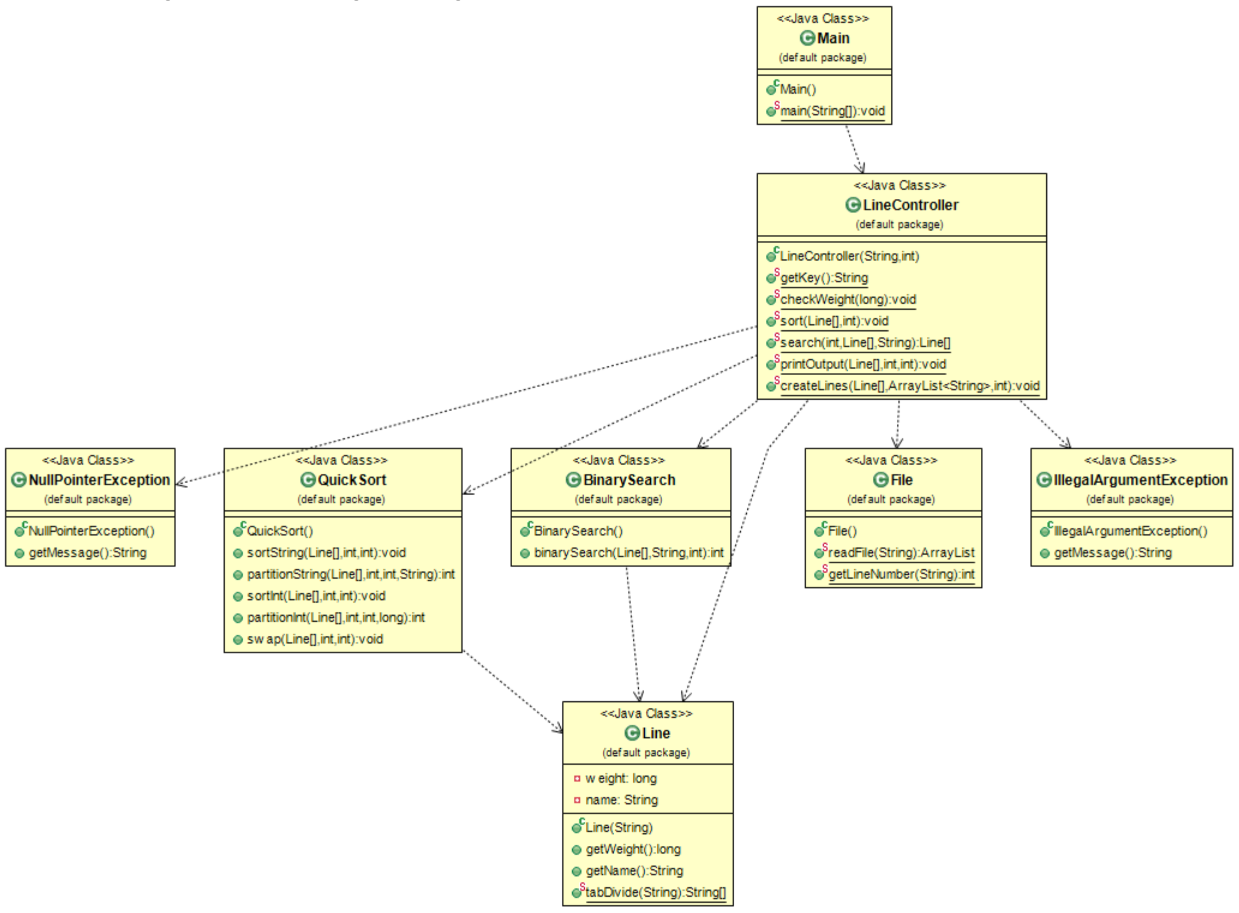
In best case:

$$NlgN \tag{2.3}$$

In the program, the above two algorithms is used. The total complexity of the program is $lgN + 2NlnN$ in average case. So the time required for this algorithm is proportional to $NlgN$ which is determined as growth rate and it is denoted as $O(NlgN)$.

2.1.5 UML diagram

The UML diagram of the designed program is as shown bellow.



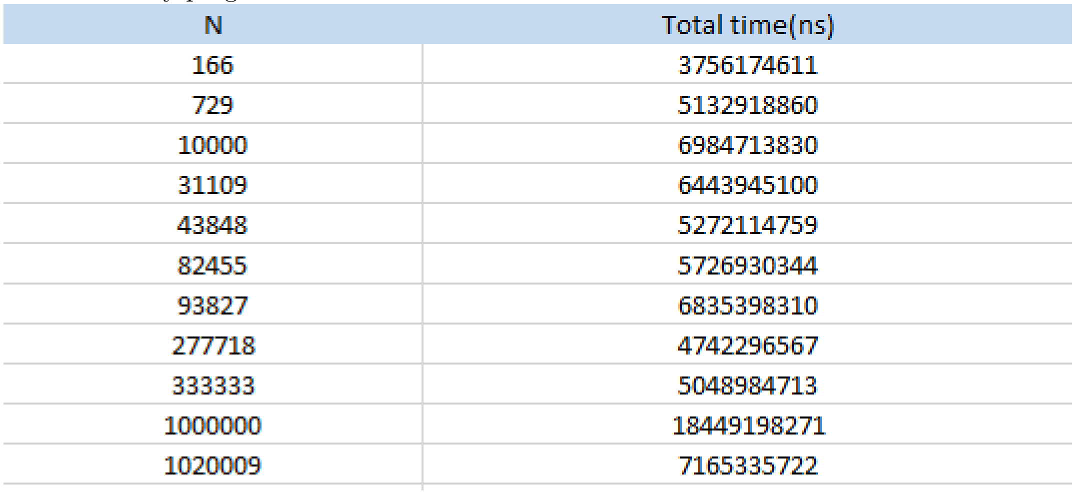
Chapter 3

Software Testing Notes

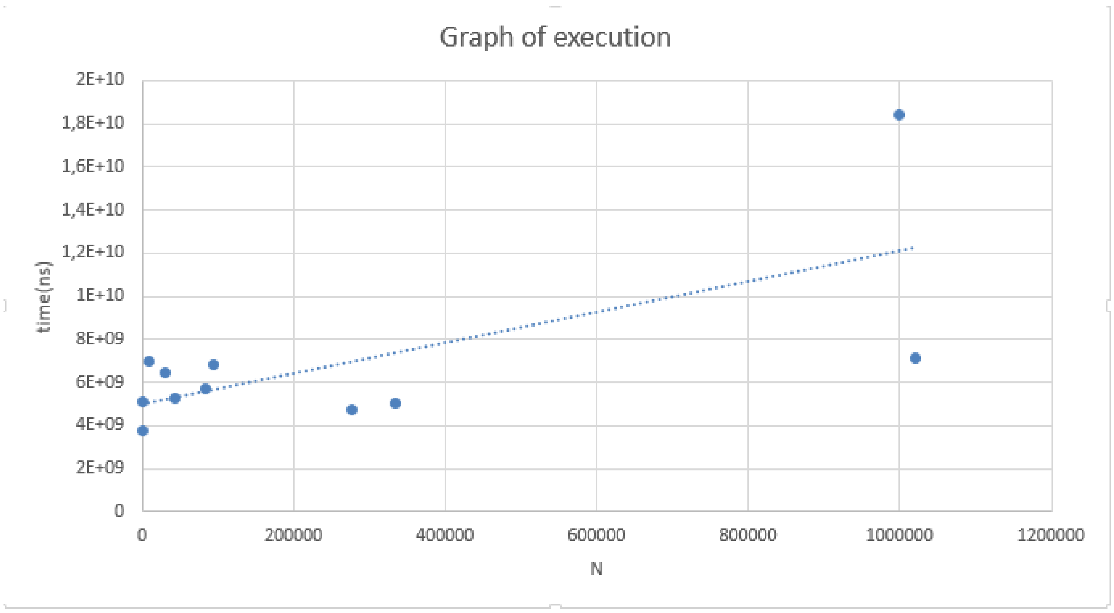
3.1 Comments

My software work really fast. It is written with Binarysearch algorithm to search the given key and Quicksort algorithm to sort the query string names and weight of it. And, for some operations has got methods (read file ,write to file, sort..) in classes. Thus, it can use easily everyplace. In addition, possible exceptions are catched by this software and it is CASE-SENSITIVE.

The table and graph show the execution time in nanosecond for different ascending n numbers in my program.



N	Total time(ns)
166	3756174611
729	5132918860
10000	6984713830
31109	6443945100
43848	5272114759
82455	5726930344
93827	6835398310
277718	4742296567
333333	5048984713
1000000	18449198271
1020009	7165335722


Graph of execution
time(ns)
2E+10
1,8E+10
1,6E+10
1,4E+10
1,2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
0
200000
400000
600000
800000
1000000
1200000
N