ELECTRONIC TELEPHONE DIRECTORY APPLICATION ANALYSIS REPORT

The electronic telephone directory system is the telephone listing of subscribers to services provided by the organisation that publishes the directory. It allows the details of the person being looked for to be found with the specified key.

The problem at hand with this electronic telephone directory is the fact that the entries appears to be random. Specifically the entries in the directory does not follow the ascending order method of name listing, hence it is very difficult to find a specific person by searching through the directory manually as this is a big datafile of about 10 000 entries. The search of a specific may be time consuming and a bit tedious because it may happen that the person in search of may be on entry 2000 or even worse than that the person might not exist.

This system is widely used in South Africa and this problem is avoided through programming.

PrintIt

public class printIt inherits BinarySearchTree. It uses the method newTree which basically creates the new BinarySearchTree of type String. The static method then reads testdata file line by line and checks if there are still entries in the file, it then trancates each entry and store the trancated entry on a varaible. The trancated entry; the last part is the full name. Under the newTree() method, the insert method is invoked which takes two parameters, in this reguard it will take the key- trancated stored entry and the full entry. The insert method then insert both the key and the full entry. The print it class has the main method which invokes newTree() method and inOrder() method which orders the entries according to fullname. The directory can then be printed out in ascending order of fullname.

SearchIt

public class SearchIt inherits BinarySearchTree. The class invokes the static method searchTree() which reads the queryfile that consists of about 20 entries of which some of these entries does not exist in testdata.

Under searchTree(), newTree() method is invoked, and uses the data that was loaded in the BinarySearchTree data structure, hence the data gets to be loaded once. SearchTree() method reads the queryfile line by line for as long as there still exists the next entry. The entry is then compared to the entry in testdata file. The find method is invoked, and takes in one paremeter, which is in the queryfile. If the name in the queryfile matches with the name in the BinarySearchTree data structure. The whole details of the entry is printed. If the entry in queryfile is not in testdata/ BinarySearchTree data structure, the output is “Not found”, implying that the entry does not exist. The SearchIt class handles the printing of output through the main method.

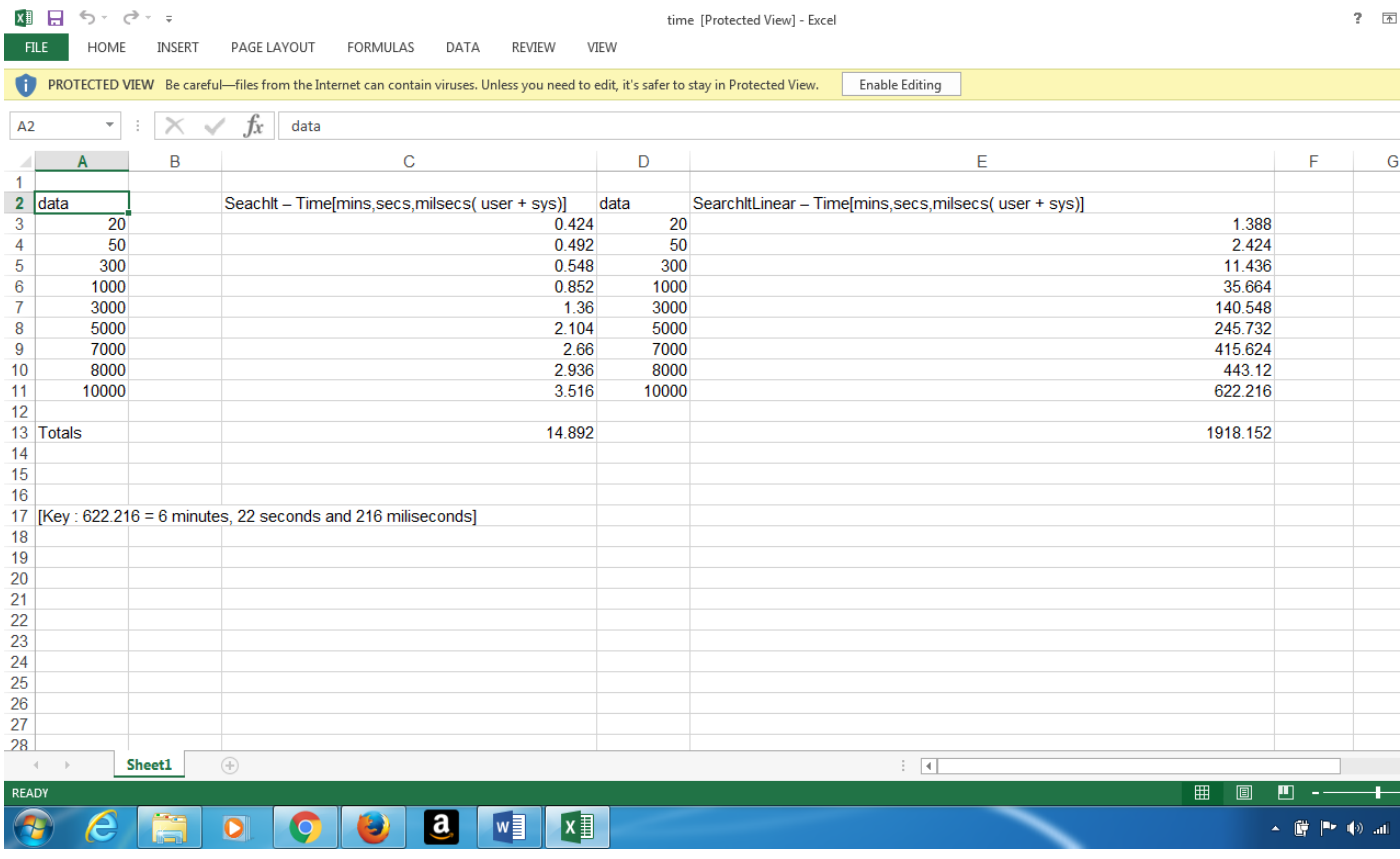
4. The first 20 lines of the output from PrintIt is in ascending order by fullname of the person. The appearance of the duplicate of a person that not matter, but what matters is

that the list will be printed in accordance of fullnames with all their datails in ascending order. The order ranges from Abbott Alec to Abernathy Cicero with all the necessary details.

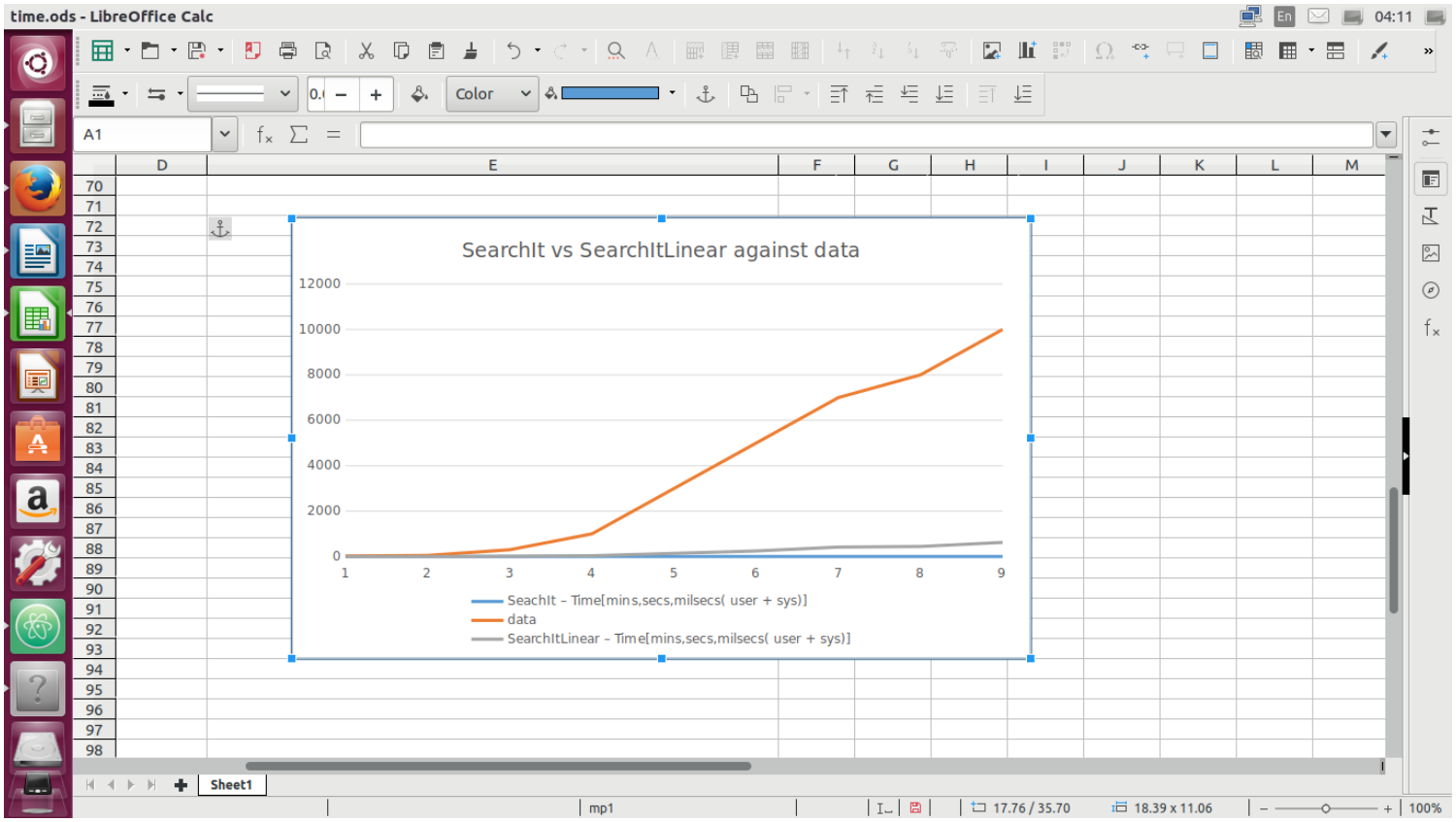
5. The queryfile consists of 20 entries ranging from Lueilwitz Candelario to Khumalo Sanelisiwe. Some of the entries in this file does not exists in testdata file. 5 of these names are not in testdata. The output from each application is the same. Both application SearchIt and SeachItLinear prints the full details of a person that is in the testdata file and prints “Not found” if the person doesn’t exist in the directory.

Attched with the report is time.ods which shows the experiment conducted where by a comparison was done between SearchIt which uses the BinarySearchTree data structure and SearchItLinear which uses the Arraylist data structure.

The entries of 20, 50, 300, 1000, 3000,5000, 7000, 8000 and 10 000 were used to conduct the experiment. Both SearchIt and SearchItLinear were run a few times to ensure faireness of the experiment. The average time it took to search for the entries from several files namely file20, file50, file300, file1000, file3000, file5000, file7000, file7000, file8000 and file10000 onto testdata file were recorded. The graph below illustates the results.



	A	B	C	D	E	F	G
1							
2	data		SeachIt – Time[mins,secs,milsecs(user + sys)]	data	SearchItLinear – Time[mins,secs,milsecs(user + sys)]		
3	20		0.424	20	1.388		
4	50		0.492	50	2.424		
5	300		0.548	300	11.436		
6	1000		0.852	1000	35.664		
7	3000		1.36	3000	140.548		
8	5000		2.104	5000	245.732		
9	7000		2.66	7000	415.624		
10	8000		2.936	8000	443.12		
11	10000		3.516	10000	622.216		
12							
13	Totals		14.892		1918.152		
14							
15							
16							
17	[Key : 622.216 = 6 minutes, 22 seconds and 216 miliseconds]						
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							



The grey curve is for SeachItLinear and the blue curve is for SearchIt. The curves illustarates SearchIt application takes less time to traverse through the BinarySearchTree in comparison with SearchItLinear. In hindsight the use of BinarySearchTree is more efficient than use of Arrays data strucures.