Project 1 Algorithms and Data Structures
Student: Hardaya Singh
ID: 692319

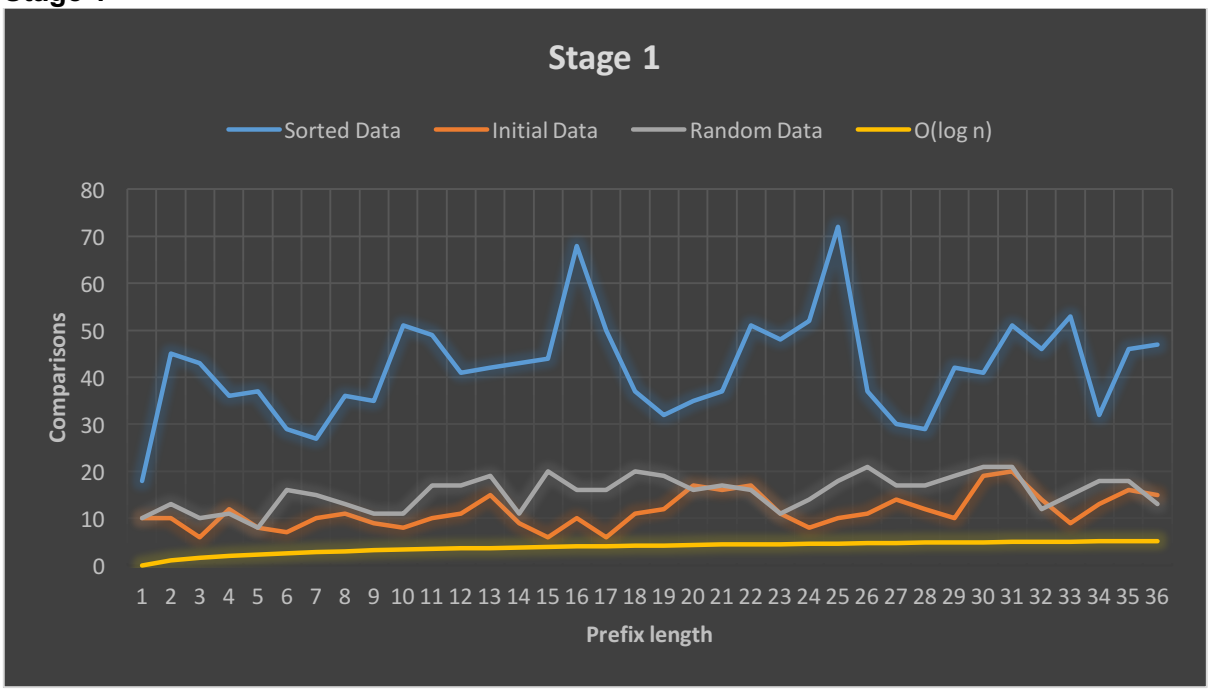
REPORT

Introduction

In this project, an autocomplete program has been designed using a ternary tree. There are two stages of the project. In the first stage the tree is filled with data. User then is asked for a prefix and the results are returned corresponding to that prefix. In the second stage, the implementation is the same but the search results are returned in a sorted order manner. Selection sort is used to sort the results and number of comparisons are reported via stdout. This report will cover the experimentation and analysis of the software using Big O theory. The program was tested against different datasets and prefixes. In stage one this reports looks at number of comparisons until the prefix is found. In stage two, the number of weight comparison it took for the sorting algorithm are reported.

Analysis

Stage 1



The graph above represents number of prefix comparisons program took against length of prefix used to search. Prefixes were randomly (but confined to that dataset) generated and then ordered in terms of length. The minimum prefix length was 2 maximum goes went up to 9.

The program behaves fairly like it should in terms of Big O theory. Since it is ternary tree, log n behaviour on average is expected. There's is also a plot of log n for reference in yellow.

When the prefix is searched from a tree in which the data is filled in a sorted manner we expect linked list behaviour and the graph shows that. Expected behaviour in terms of search O(n) when the tree is filled in a sorted manner. Here we see very random behaviour but it is definitely worse than the other two methods. When the prefix is searched from dataset where the tree is filled in a random order. We can almost expect log n behaviour of our search engine. The program behaves fairly like so. The only difference is that it is

escalated a little. It could be because smallest prefix consisted of two letters not zero. The escalation of the graph can also be explained by due to continuous searching.

The search engine can be improved drastically if BST or AVL tree is used. It would guarantee the tree staying balanced as it is being filled.

Stage 2

The graph speaks for itself. The n^2 behaviour can be seen as expected with the theory. The graph plots number of prefix results found against comparisons made by the selections sort. A linked list has been used to get data prefix results from the ternary tree. So selection sort works well in terms of implementation with the linked list. Most other algorithm would require to know the size of the data. Here linked list is used because number of results that will be returned are unknown. Therefore, in selection sort all we need is the head pointer and keep moving the head as we sort.

The sorting can easily be improved by using a better search algorithm ie merge sort or quick sort. The only difficulty with those would be the implementation. Therefore linked list together with selection sort works well dynamically.

Conclusion

In this project a ternary tree was implemented. And then it was used make an autocomplete program that took a string input and returns all the keys matching that prefix in the database. The software was then tested with various datasets of different sizes. It showed that it can work efficiently with almost log n behaviour on average case.
In stage 2 the prefix results were returned in sorted manner in terms of their weight. The algorithm used were selection sort but the results can be improved by using a better algorithm. This software can be used in various real life applications and help save users a lot of time