Exercise 1: Set Implementations Report

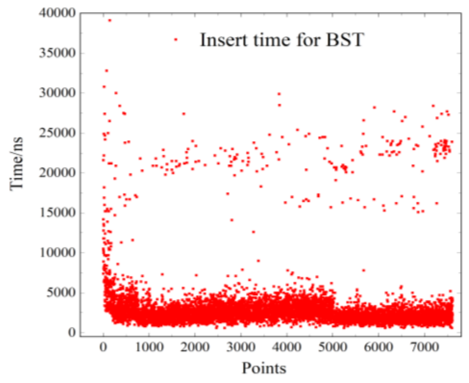
Yilin Wang

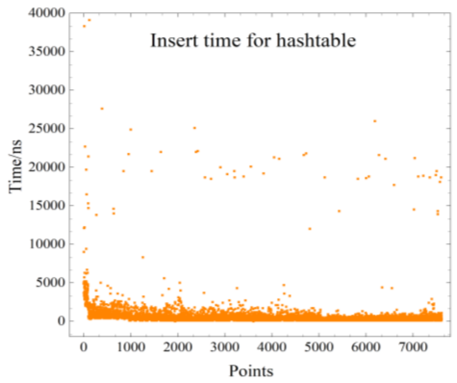
Part 1 – Insert to the set. All of the sets return 7611 words, and I treat "_you_" and "you" as the same word.

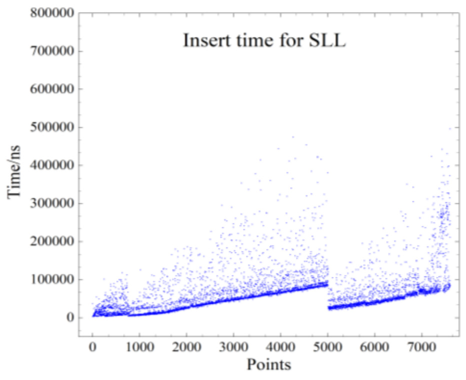
BST: The runtime for add(), contains() operation is $\Theta(\log N)$, and $\Theta(1)$ for size().

HashTable: The runtime for add(), contains() operation is $\Theta(\mathrm{K})$, where K is the number of items with the same hash code. Because the total buckets number will be doubled when $N / M > \text{Load Factor}$, K would not be very large. And $\Theta(1)$ for size().

Singly Linked List: The runtime for add(),contains() operation is $\Theta(\mathrm{N})$, and the size() operation is $\Theta(1)$.







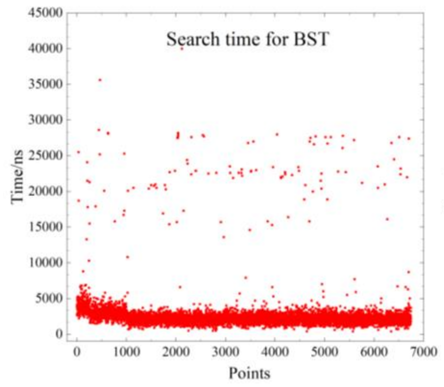
Part 2 – Search the sets. All of the sets returned 112 words that are not in the set.

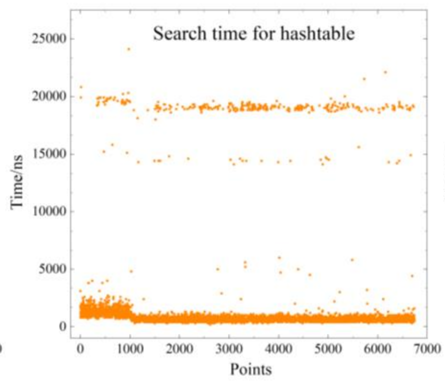
BST: The worst case is when we keep adding right (or left) children to the tree, and in this case, the runtime is $\Theta(\mathrm{N})$. The best case is when the tree gets really bushy, and the runtime is $\Theta(\log N)$. The average case is that it's unlikely to add only one side node to the tree, so in average, the runtime is $\Theta(\log N)$.

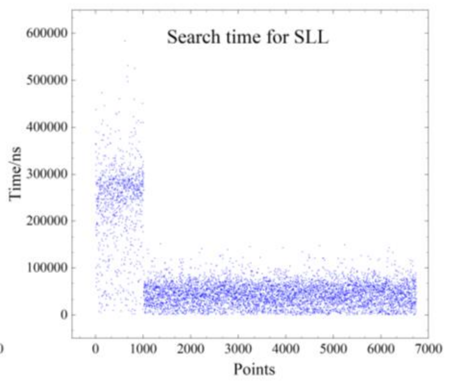
Hash Table: The worst case is when all of our strings end up with the same hash code, then our hash table could be a singly linked list. The runtime for that would be $\Theta(\mathrm{N})$. The best case is when every string has a unique hash code, then the runtime would be $\Theta(1)$, but this could cause $\Theta(\mathrm{N})$ space complexity. The average code is when we use a linked list for the same hash code strings. And the runtime would be $\Theta(\mathrm{K})$.

SLL: The worst case is to travel all of our strings, and the runtime is $\Theta(\mathrm{N})$. The best case is when we find our target at first try, which is $\Theta(1)$. The average case would be $\Theta(\mathrm{N})$ as well.

As showed on the figure, the search time for SLL is the slowest and hash table is the fastest, which are expected in the analysis. And for BST, sometimes search is slow, and that's because the target is deep in the tree. And for hash table, most of the search is fast, only a small portion is slow. They would probably has the same hash map, and they ended up in a relatively long linked list. SLL is the slowest for sure, and most of search need to travel a large portion of the list. And all of three perform a relatively slow search at first, this could be due to the front part of words lands in the ending part of our data structure.


Search time for BST
Time/ns
Points


Search time for hashtable
Time/ns
Points


Search time for SLL
Time/ns
Points