# Big O Analysis

**Unordered ArrayList**

*exactMatch(): O(n)*

Worst case exactMatch() has to look through the whole list to find the word.

*allMatches(): O(n)*

Has to traverse the whole list to find every match.

**Unordered LinkedList**

*exactMatch(): $O(n^2)$*

exactMatch() loops through the whole list which is O(n) and get() for a LinkedList is also O(n) contained in the loop.

*allMatches(): $O(n^2)$*

Loops through the whole list and uses get() in the loop which is an *n* operation *n* times.

**Ordered ArrayList**

*exactMatch(): O(log(n))*

Binary Search starts in middle of last forcing it to be balanced guaranteeing log(n) performance.

*allMatches(): O(n)*

Goes through the whole list to find all matches same as Unordered.

**Ordered LinkedList**

*exactMatch(): O(log(n))*

Binary Search starts in middle of last forcing it to be balanced guaranteeing log(n) performance.

*allMatches(): $O(n^2)$*

Uses get() in a loop which is an *n* operation *n* times.

**Binary Search Tree**

*exactMatch(): O(n)*

Doesn't guarantee a balanced tree so worst case is you have to traverse every single node.

*allMatches(): O(n)*

Traverses the whole tree which is an *n* operation

**Trie**

*exactMatch(): O(1)*

The time complexity is based on the amount of characters in the target and not the number of words in the trie.

*allMatches(): O(n)*

Traverses the whole trie which is based on how many words are in the trie.

**HashTable**

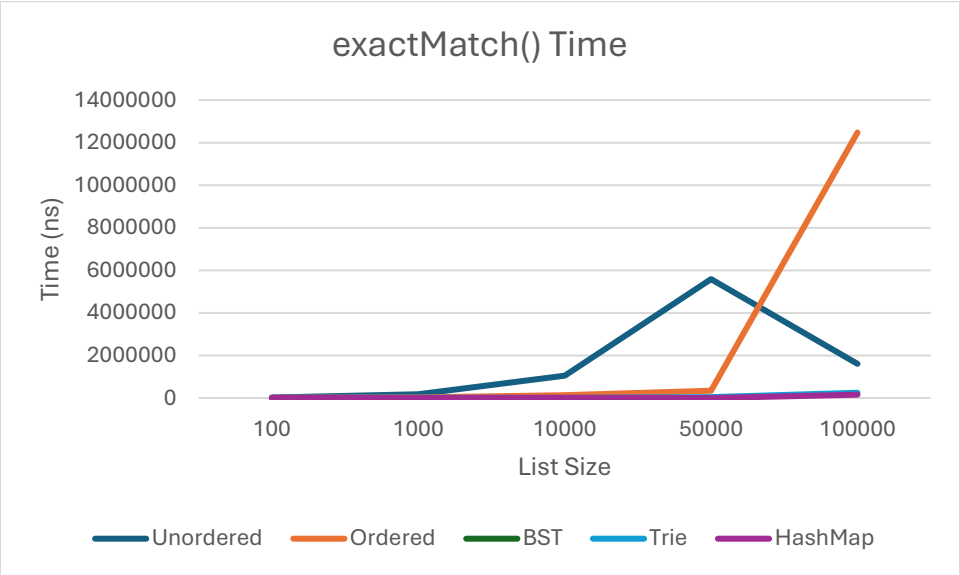
*exactMatch(): O(1)*

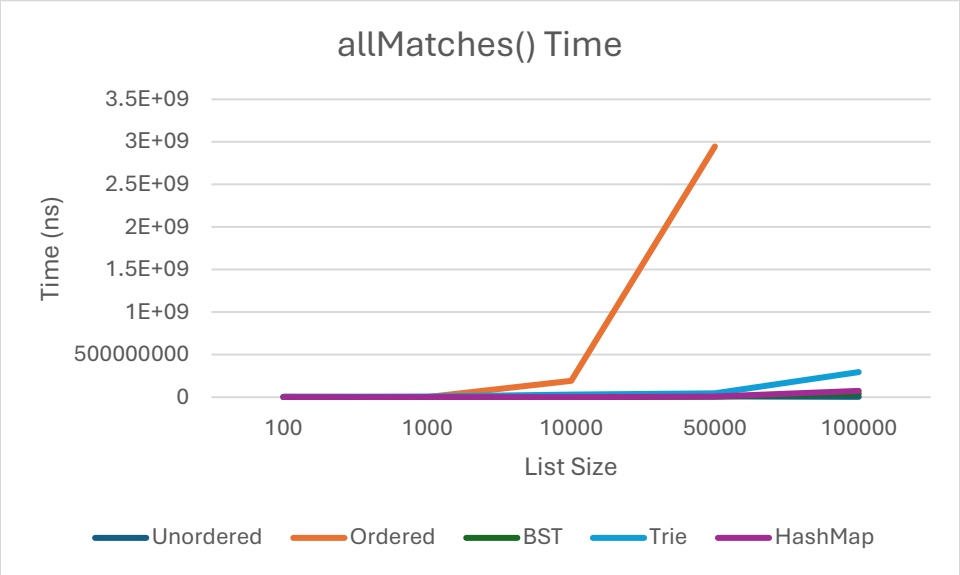
Uses the contains() method of the HashSet which is constant time.

*allMatches(): O(n)*

Traverses the whole HashSet to find every match.

# Benchmarking




*Note: Ordered LinkedList 100000 was removed for graph scale*

The number of strings was chosen that allowed for a reasonable curve on the graph as well as getting to larger values of *n*. Data gathering could be improved with longer amounts of time to benchmark and a dedicated device to benchmark on. The extra benchmarking would allow for averaging of data and removal of outliers. This can be done with smaller values of *n* but is difficult to do with large values of *n* on our laptops.

## Reflection

In order to better understand this project, I would try to better understand how a trie works to better implement it. I would also try to learn how better to optimize adding and getting from random places in a LinkedList. My greatest strength with this course is that I can pick up on most topics fairly quickly. This then makes labs and homework easier. I believe that an area I could improve in is my code efficiency. This is something that can be improved with more experience but I also don't always use the most efficient method for a given task because it's easier to implement. A software project that I could implement using what I have learned in this course is some kind of cataloging software. I am interested in model railroading and from a quick search there don't seem to be many options for cataloging models. Many collectors have hundreds of models, and a simple cataloging software would be niche but very valuable within the community.