Search Algorithm Comparision

Avula Ankith Reddy
1002090721
Sravan Chandaka
1002059166

Department of Computer Science and Engineering
University of Texas Arlington

December 2, 2022

Introduction

- Search Algorithms are used to search for elements in a given list or array.

Linear Search

- Linear search is a search algorithm where an element is searched by traversing through the array of n elements and comparing each element of the array
- If the element is found in the array then the index value of the element is returned. else -1 is returned as output of algorithm
- The Time Complexity of the Algorithms is O(n)

Binary Search

- Binary Search algorithm is implemented when the given search array is a sorted array of n elements
- The element which is to be searched is compared to the median of the array and then accordingly the comparison goes to left or right half of the array depending on whether the element is smaller or larger than the element
- The Time Complexity of this algorithm is O(log(n))

Binary Search Tree

- Binary Search Tree is a data structure made from an array of n elements. The tree has nodes with at most two leaves.
- The values on left side of the parent node are smaller or equal to value of the parent node. The values on the right side of parent node are greater than the value in the parent node.
- The algorithm does not require a sorted array.
- The Time Complexity of this algorithm is O(log(n))

Red Black Tree

- A Red-Black Tree is a variant of binary tree where it follows certain properties to have it height maintained so that the worst-case scenario search complexity is always O(log(n)).
- The Red-Black Tree has nodes of two colors red and black. The root node is always colored red and leaf nodes including the NULL nodes are colored black.
- The nodes are colored in a manner where two red nodes are not found consecutively.
- The Time Complexity of this algorithm is O(log(n))

Time Complexity of Algorithms

-	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)
Binary Search Tree	O(1)	O(log n) or O(h)	O(n)
Red Black Tree	O(1)	O(log n) or O(h)	O(log n) or O(h)

* h is the height of the tree

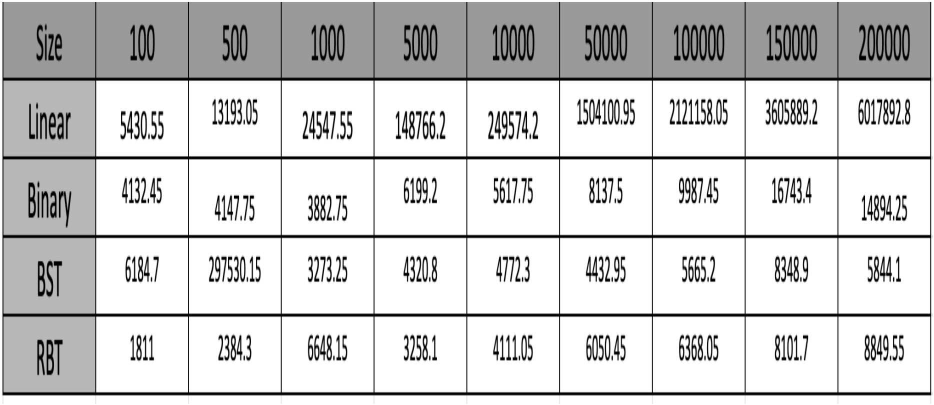
* n is the number of elements in the array

Implementation

- We used a Random integer generator to randomly generate the values between 0 to 1000000.
- We used a condition, that the key should definitely be present in the array index to be searched.
- Execution times of different searching techniques is calculated excluding the sorting time calculation for the Binary search, BST, and RBT building time of the tree.
- We considered different array sizes ranging from 100 to 200000.
- By performing the randomized searching techniques 20 times subsequently, we were able to show the best possible Average case scenario of all four searching algorithms.
- We plotted the runtimes using matplotlib library.

Results: Runtimes

The following table shows time taken by each algorithm for searching for different data sizes in nanoseconds(ns)



Size	100	500	1000	5000	10000	50000	100000	150000	200000
Linear	5430.55	13193.05	24547.55	148766.2	249574.2	1504100.95	2121158.05	3605889.2	6017892.8
Binary	4132.45	4147.75	3882.75	6199.2	5617.75	8137.5	9987.45	16743.4	14894.25
BST	6184.7	297530.15	3273.25	4320.8	4772.3	4432.95	5665.2	8348.9	5844.1
RBT	1811	2384.3	6648.15	3258.1	4111.05	6050.45	6368.05	8101.7	8849.55

Results: Comparison

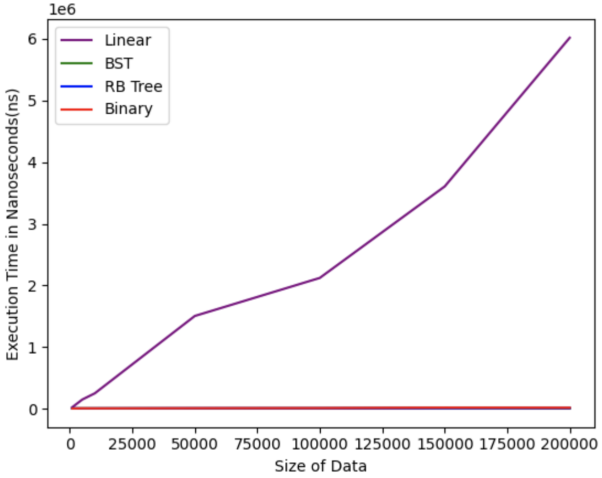


Figure: Plot between all algorithms

Results: Comparison

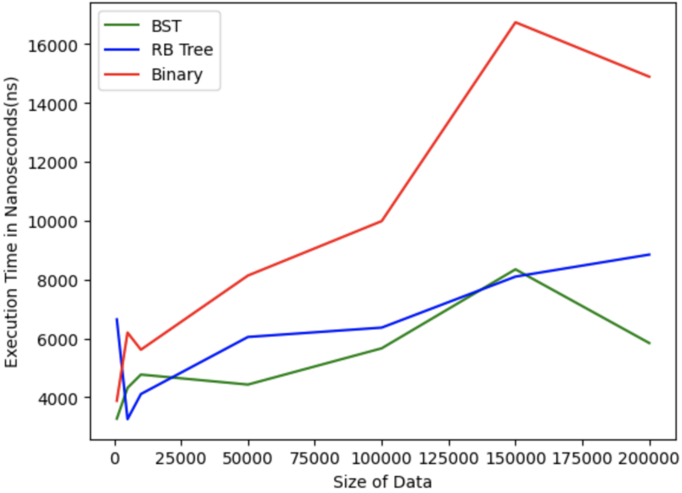


Figure: Plot between Binary, BST, RB Tree

Observations

- The best performance is observed in Red Black Trees followed by Binary Search Trees and Binary Search Algorithms.
- For a sorted Array using a Binary search is better since we do not need to build a data structure and we save space.
- For an unsorted Array, Binary Search Trees and Red Black Trees perform better but comes with an additional space complexity O(n).

Improvements

- A linear search has a time complexity of O(n), we can try a prune and search algorithm where it prunes and searches the array for the given element using the partition algorithm in quicksort. But it also yields a time complexity of O(n).
- For a sorted and uniformly distributed array as input the best algorithm we can use is **interpolation search** since it provides the best case time complexity O(log log(n)) and shorter time duration but its worst case time complexity goes till O(n)
- A Binary Search Tree in the worst-case scenario yields a linked list of n nodes where search complexity becomes O(n) this is the case where we implement balanced tree algorithms to maintain tree height on both sides like **Red Black Tree**, **AVL Tree**, **2-3-4 Tree**, etc.
- **AVL trees** provide faster lookups than Red-Black Trees because they are more strictly balanced

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- https://www.programiz.com/dsa/red-black-tree
- https://www.geeksforgeeks.org/g-fact-84/
- https://en.wikipedia.org/wiki/Interpolation_search
- `https://www.geeksforgeeks.org/red-black-tree-vs-avl-tree/#:~:text=AVL%20trees%20provide%20faster%20lookups,they%20are%20more%20strictly%20balanced.&text=In%20this%2C%20the%20color%20of,is%20either%20Red%20or%20Black.`