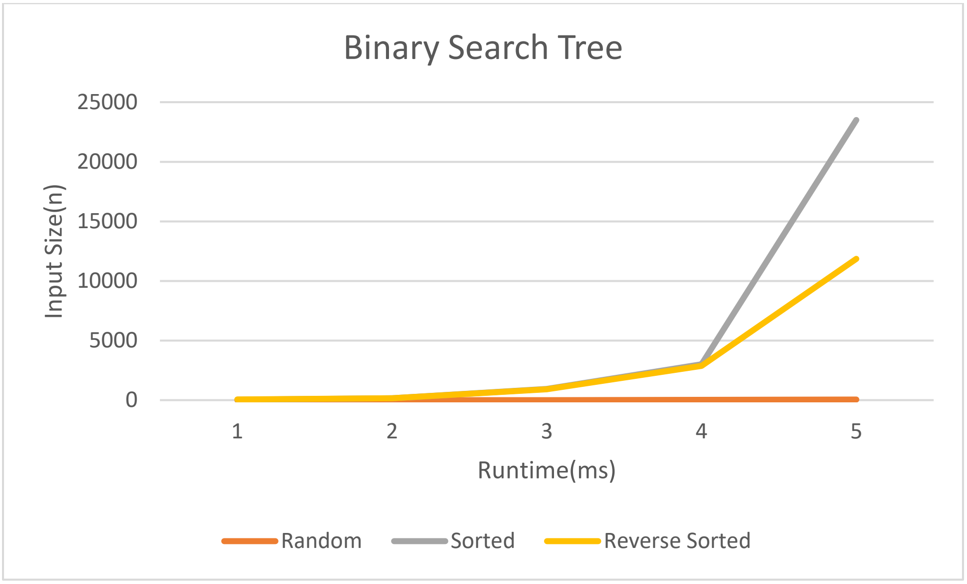
CS: 590 Algorithms

Binary Search and Red-Black Trees

Poorvi Raut: 20009560

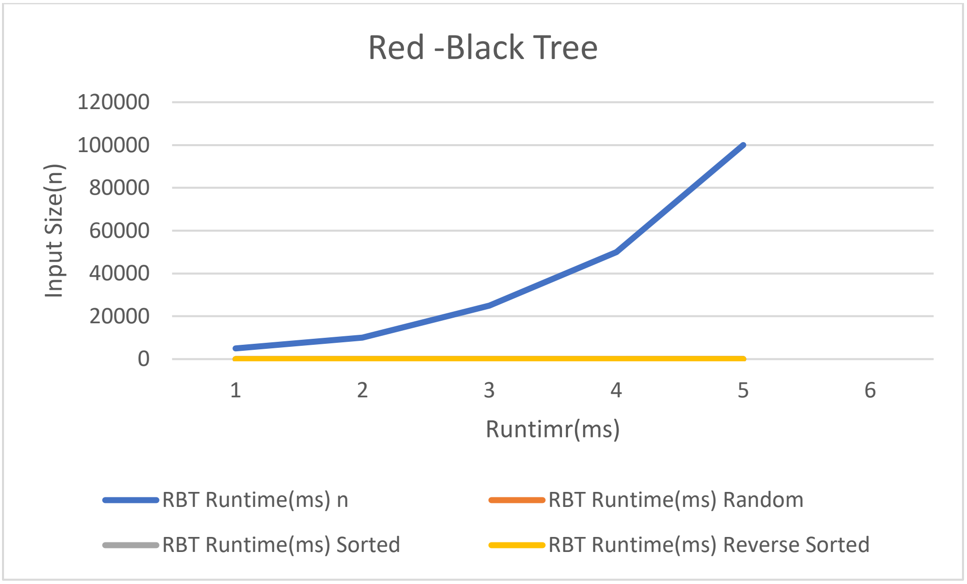
Binary- Search Tree

BST Runtime(ms)			
n	**Random**	**Sorted**	**Reverse Sorted**
5000	1.3	37.7	54.5
10000	2.4	149.1	145.9
25000	7.7	935.7	909.3
50000	18.1	3003.3	2875.8
100000	47.6	23504.1	11854.8



Red-Black Tree

RBT Runtime(ms)			
n	Random	Sorted	Reverse Sorted
5000	1.9	1.0	1.1
10000	3.6	2.2	2.1
25000	9.2	7.2	6.0
50000	23.2	16.6	14.5
100000	44.5	34.2	29.5



COUNTERS

Counter when input is Sorted								
n	RBT							BST
	Black Height	Case 1	Case 2	Case 3	Left Rotate	Right Rotate	Duplicates RBT Values	Duplicates BST Values
5000	11	0	0	0	0	0	0	0
10000	12	0	0	0	0	0	0	0
25000	14	0	0	0	0	0	0	0
50000	15	0	0	0	0	0	0	0
100000	16	0	0	0	0	0	0	0

Counter when input is Reverse Sorted								
n	RBT							BST
	Black Height	Case 1	Case 2	Case 3	Left Rotate	Right Rotate	Duplicates RBT Values	Duplicates BST Values
5000	11	0	0	0	0	0	0	0
10000	12	0	0	0	0	0	0	0
25000	14	0	0	0	0	0	0	0
50000	15	0	0	0	0	0	0	0
100000	16	0	0	0	0	0	0	0

Counter when input is Random Sorted								
n	RBT							BST
	Black Height	Case 1	Case 2	Case 3	Left Rotate	Right Rotate	Duplicates RBT Values	Duplicates BST Values
5000	8	0	0	0	0	0	0	0
10000	8	0	0	0	0	0	0	0
25000	9	0	0	0	0	0	0	0
50000	10	0	0	0	0	0	0	1
100000	11	0	0	0	0	0	0	3

OBSERVATIONS:

Discussing and comparing runtime of Binary Search Tree and Red Black Tree

1. The BST insert tree algorithm traverses the tree from top-down approach i.e., from root to leaf node to find the location where the insertion of node is required. At every level each node is visited, and comparison is made such that the BST rules are not violated. A BST with 'n' nodes has log n levels at most 'n' levels. Therefore, the runtime complexity for insertion of node for best case and worst case is **O (log n)**.
2. **BST Tree** becomes unbalanced when sorted in ascending and descending order for inserting a node at any level. The inorder algorithm walks 'n' nodes and hence the runtime complexity becomes **O(n)**
3. Inserting elements in random order makes the tree remain balanced, hence runtime complexity for random sort of BST is **O (log n)**.
4. **RBT** is balanced Tree therefore for random and sorted arrays the runtime complexity remains **O (log n)**. Whereas the sorted and reversely sorted BST runtime increases exponentially as the input size increases.

In the table with counter comparisons of BST and RBT:

• When array is reversely sorted - there are no occurrence of Case 2 and left rotation for RBT. Since the array is reversely sorted, case 1 and case 3 chances of occurrences are more as it involves left subtree.

• Similarly, when array is sorted – there are no occurrences of case 2 and right rotation.

•There are no duplicate values because the value of ‘n’ is very small.

Black height test shows output in RBT, recording the black height when test succeeds – thus testing the Red-black property 5 in RBT.

• Every path from the root to any of the leaf nodes must have the same number of black nodes