CS4522 Advanced Algorithms – Assignment 01
B.K.D.V.Vimarshana – 140640A

1.

B-TREE-SEARCH time = O(h*T) where,

h = height of the B-TREE

T = time taken for binary search within each node

Here,

h = $O(\log_t n)$ [1]

Since number of keys in each node is less than 2t-1,

T = O(lg t)

Therefore,

B-TREE-SEARCH time = $O(\log_t n * \lg t)$

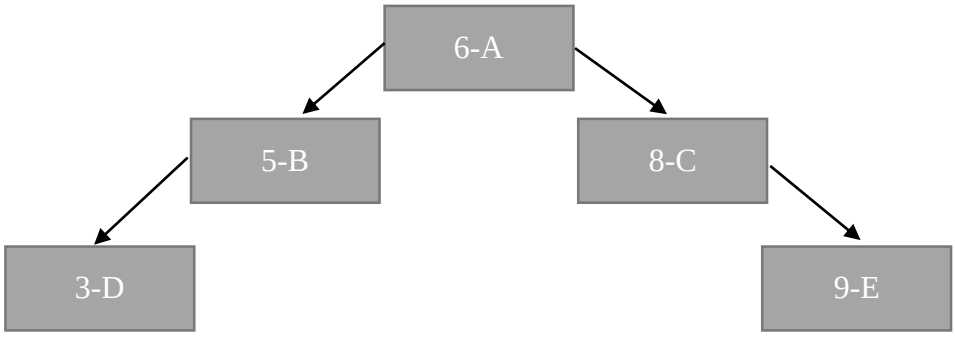
$= O((\lg n / \lg t) * \lg t)$

$= O(\lg n)$

2.

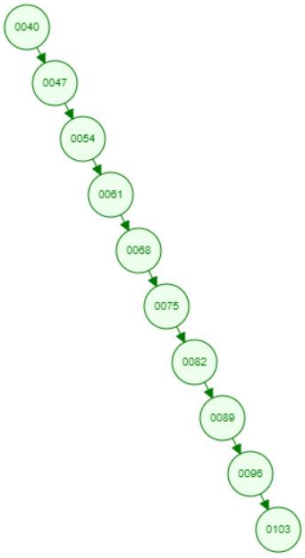
Suppose priorities of the keys as follows which A has the maximum priority and E has the minimum priority

6 → A
5 → B
8 → C
3 → D
9 → E

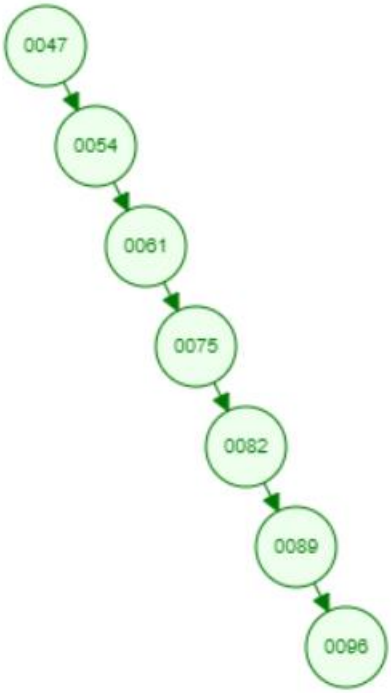


3.

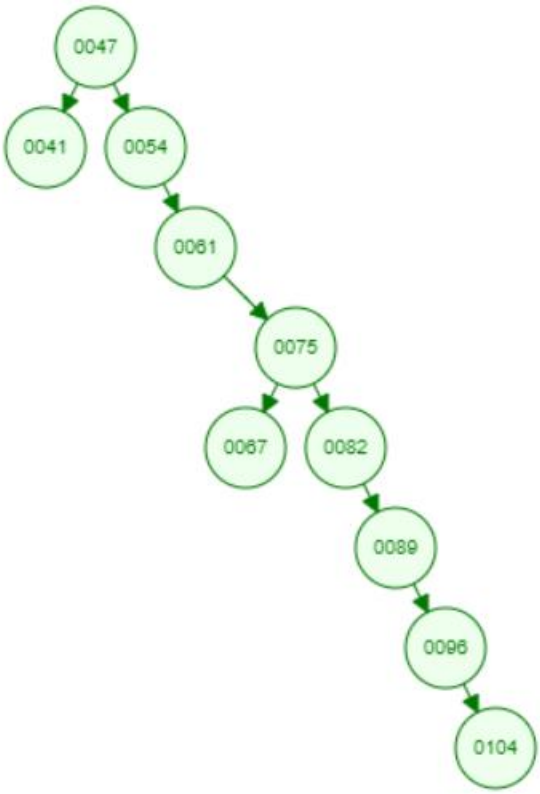
a) i) Insert: 40, 47, 54, 61, 68, 75, 82, 89, 96, 103



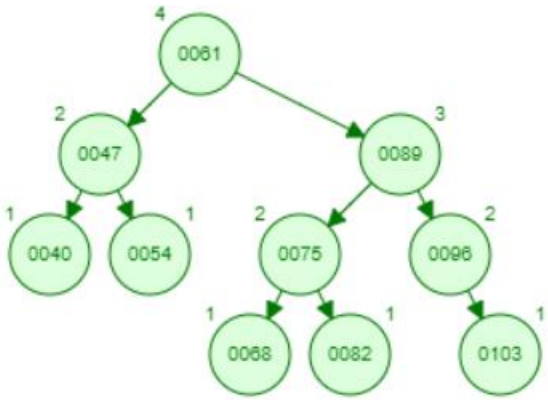
a) ii) Delete: 40, 68, 103



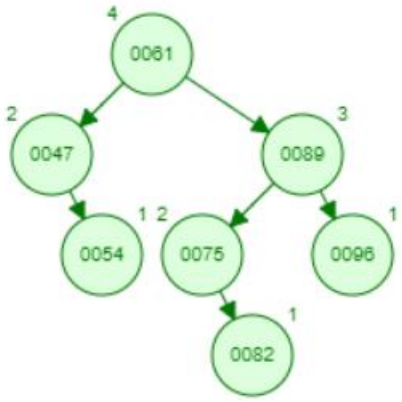
a) iii) Insert: 104, 67, 41



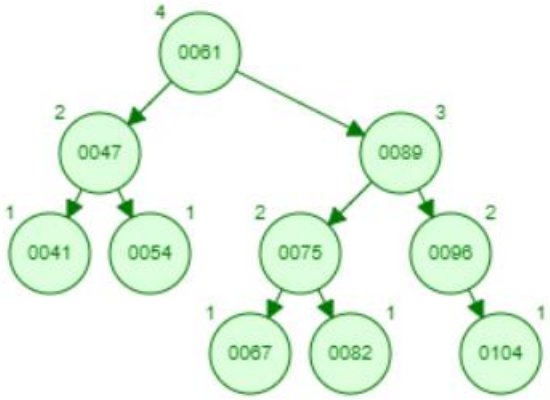
b) i) Insert: 40, 47, 54, 61, 68, 75, 82, 89, 96, 103



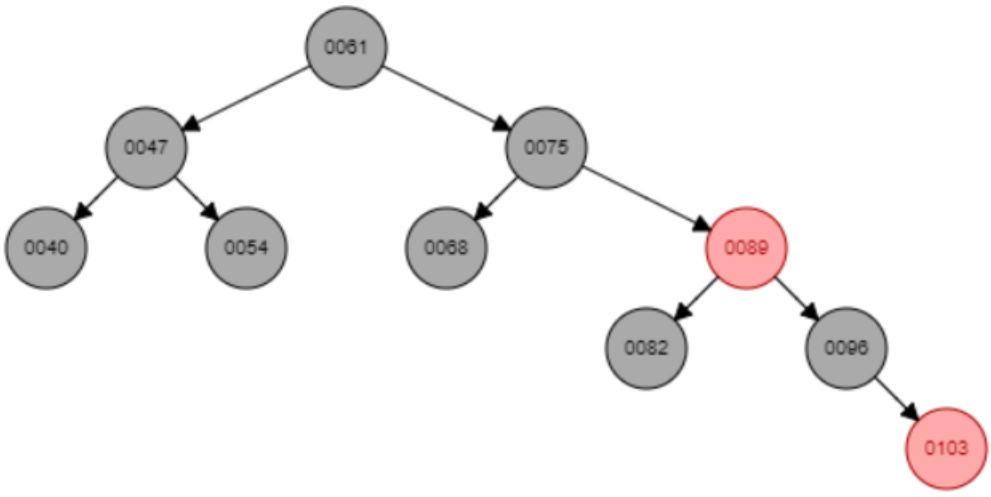
ii) Delete: 40, 68, 103



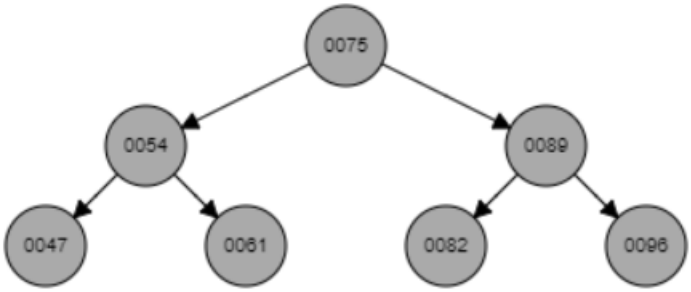
iii) Insert: 104, 67, 41



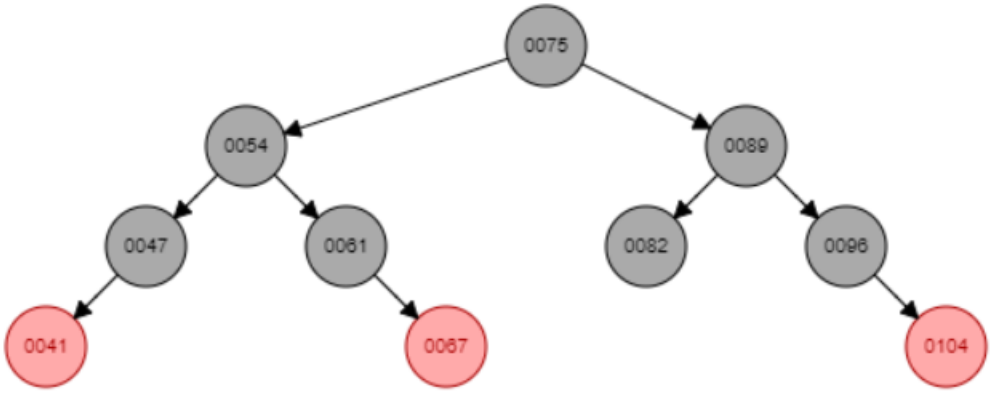
c) i) Insert: 40, 47, 54, 61, 68, 75, 82, 89, 96, 103



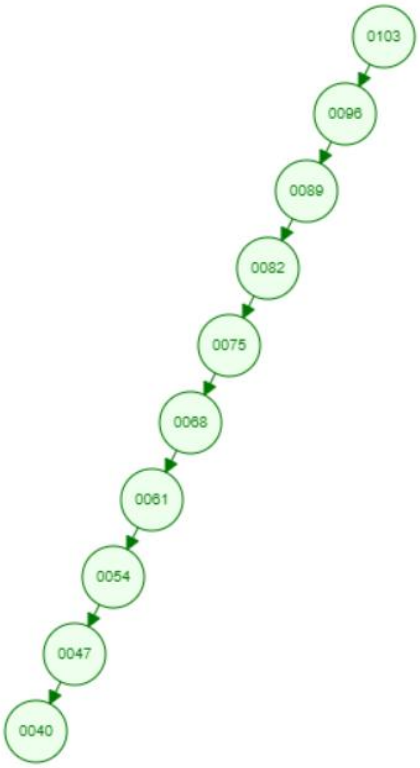
ii) Delete: 40, 68, 103



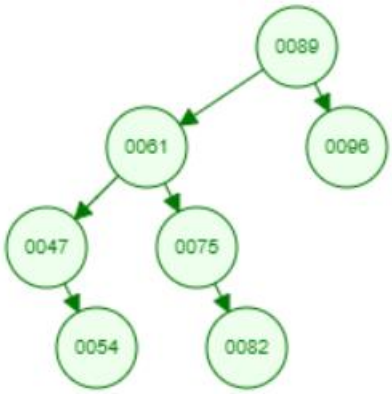
iii) Insert: 104, 67, 41



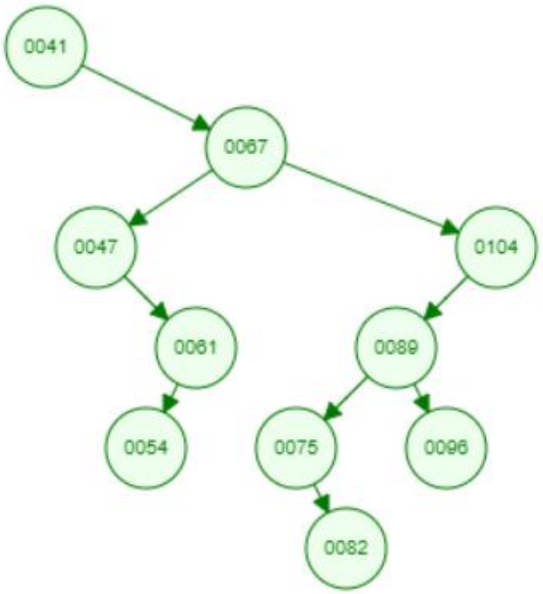
d) i) Insert: 40, 47, 54, 61, 68, 75, 82, 89, 96, 103



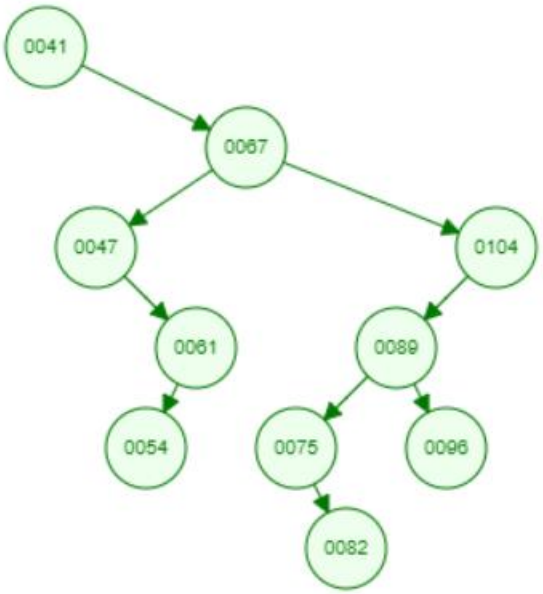
ii) Delete: 40, 68, 103



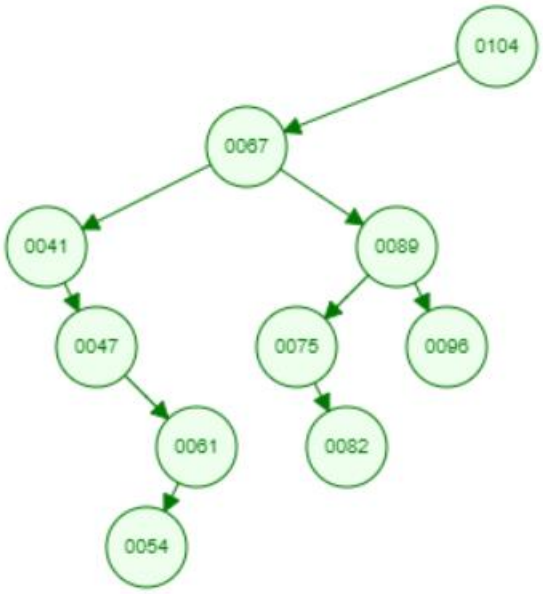
iii) Insert: 104, 67, 41



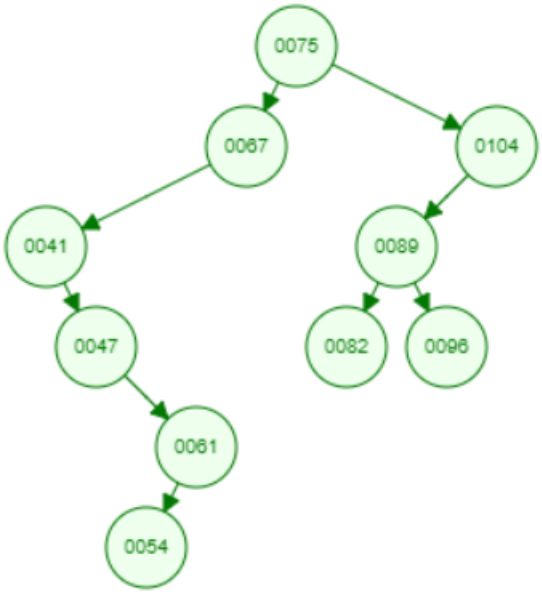
iv) Search 41



v) Search 104



vi) Search 70



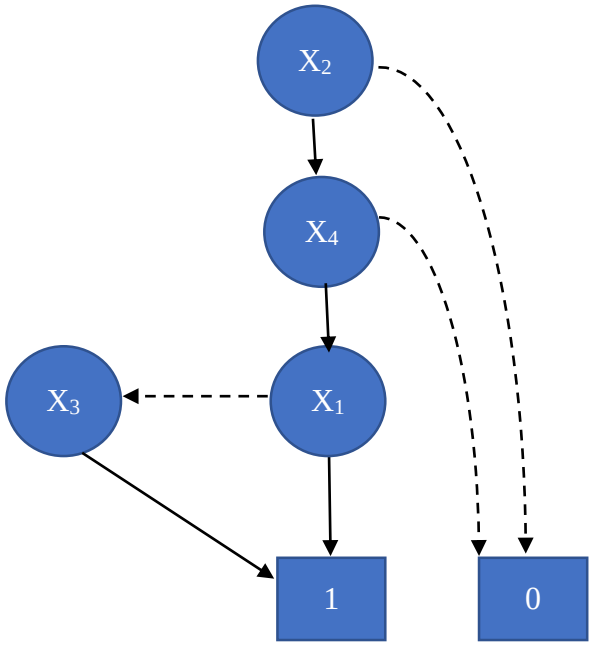
e) Here the balance of the BST depend on the inserting order. In this example BST is unbalanced. But in AVL-Tree and Red-Black tree irrespective of inserting order tree becomes balanced due to the rotation. Even though splay-tree also used rotation it could not be balanced as splay-tree used rotation to bring the recently accessed node to the root.

4.

(a) $f = x_1x_2x_4 + \bar{x}_1x_2x_3x_4$

$g = x_2x_4$

(b)



(c)

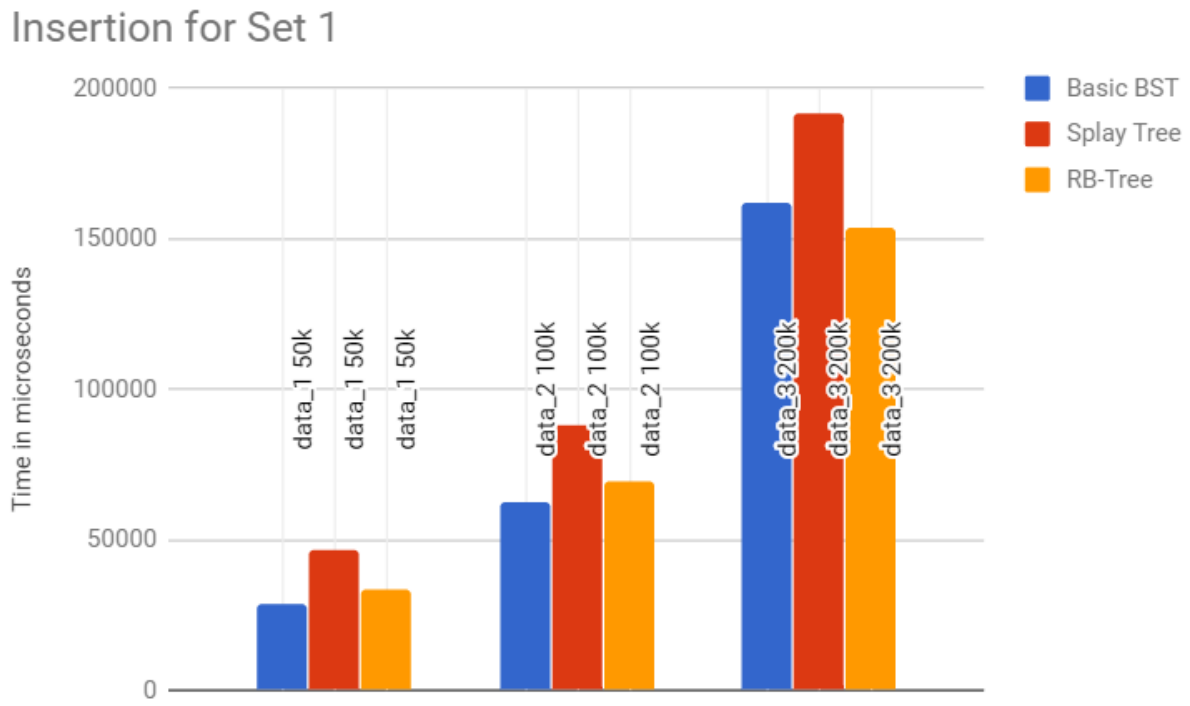
- Complexity of the most of the operations in BDD is depend on the size of the BDD.
- From the above example it is clear that by changing the variable order size of the BDD can be reduced. Hence, Complexity of the operation in BDD depend on the variable ordering. This is called ‘variable ordering problem in BDDs’.

5.

(a)

Time taken to **Insert** (in Microseconds)					Data set size
Set 1	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	28346	46493	33463	50000
	data_2	62352	87905	69608	100000
	data_3	162218	191728	153365	200000

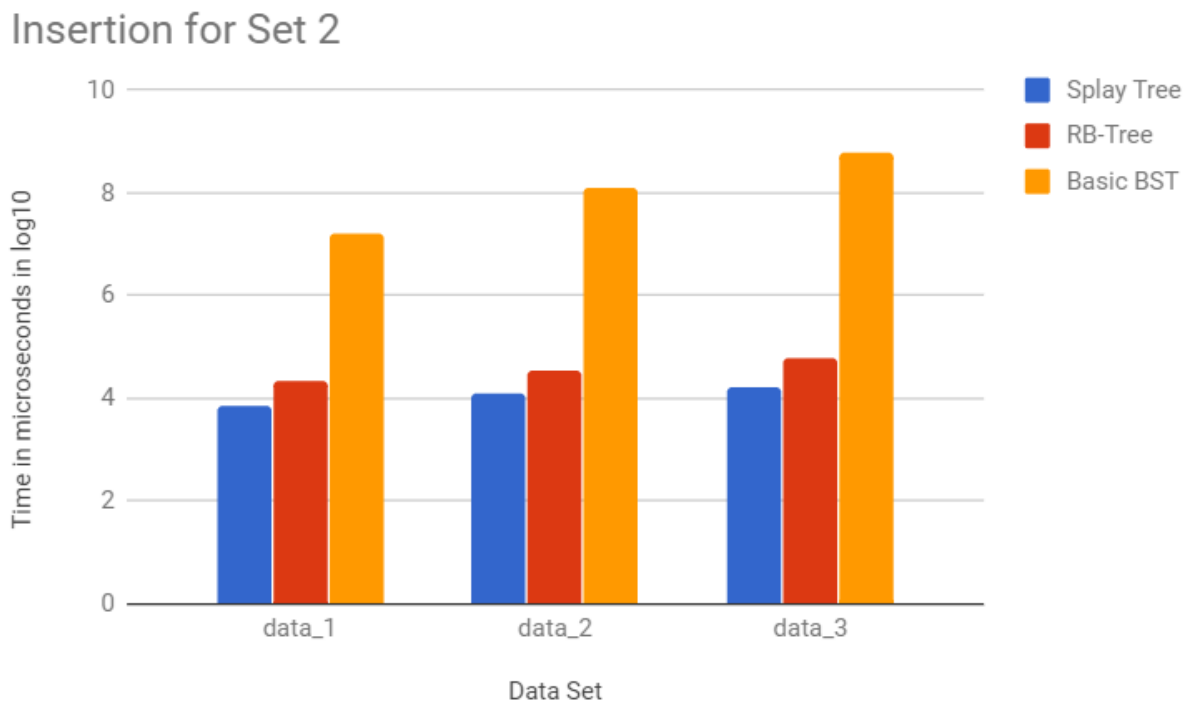
Following graph shows the time taken for Set 1 to complete insertions of 50000, 100000 and 200000 items respectively to Basic BST, Splay Tree and RB-Tree.



It is clear that when the size of the data set increases time taken to complete all the insets also increases for all Trees. Out of three Trees Splay Tree has got the maximum time to complete the task. The reason for that would be Splay Trees do the rotation as a part of the insert operation to keep the latest accessed node in the root which is the node that inserted. In dataset 1 and dataset 2 time taken by Basic BST is less than RB-Tree since RB-Tree uses rotation operation. But with the size of the Tree increases RB-Tree get balance. Therefore time taken to insert an item decreases as the height of the RB-Tree is less than Basic BST.

Time taken to **Insert** (in Microseconds)					Data set size
Set 2	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	16502823	7395	21768	50000
	data_2	120492447	12333	35360	100000
	data_3	596242638	16829	59615	200000

Following graph shows the time taken for Set 2 to complete insertions of 50000, 100000 and 200000 items respectively to Basic BST, Splay Tree and RB-Tree. Specialty of Set 2 is numbers in Set 2 has sorted.



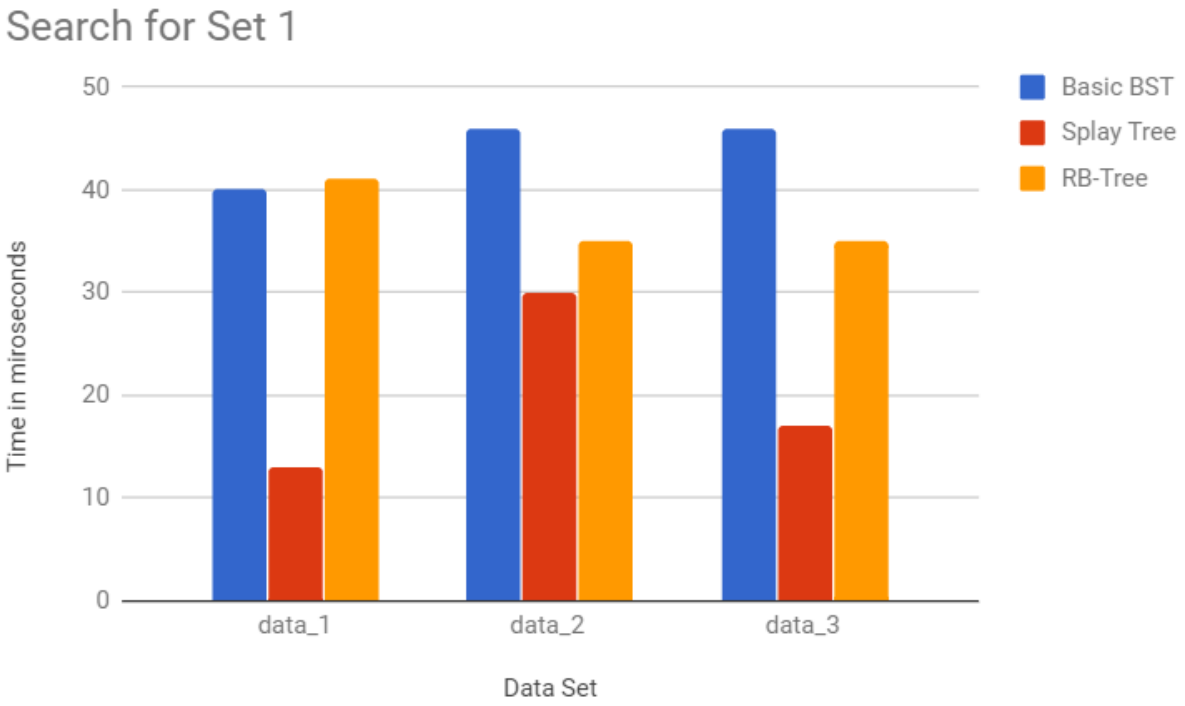
Time taken for complete the task for Basic BST is so high compared to RB-Tree and Splay Tree. This is because the data set is sorted. In Basic BST when inserting an item the key of the node

will be compared with the key values of each key and finally added to the Tree as a leaf. Then time taken to insert an item to the tree increases with the growth of the Tree. Here time complexity of the insertion is the worst case scenario. This is the same reason to increase the time taken to complete the task exponentially when the size of the dataset increases.
By looking at the table we can observe that Splay Tree has less time compared to RB-Tree. Reason for this is the specialty of the dataset. Since in Splay Tree the recently added item is in the root the item that is going to insert will be added as a child node to the root. In RB-Tree some time has to spend to find the node to add the item.

b)

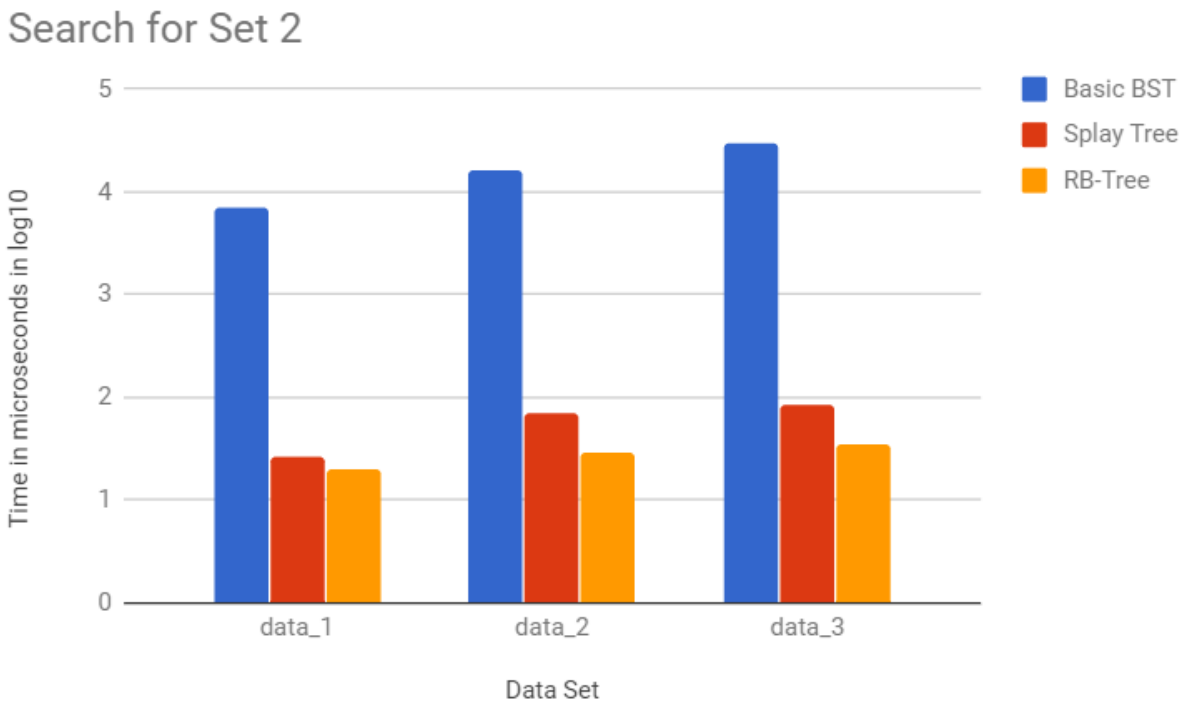
Time taken to **Search** (in Microseconds)					Data set size
Set 1	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	40	13	41	10
	data_2	46	30	35	10
	data_3	46	17	35	10

Following graph shows the time taken for Set 1 to complete search of 10 items in Basic BST, Splay Tree and RB-Tree. Specialty of these 10 items is they are the second last 10 items in the insert datasets. Hence, these items are recently added items to each Tree.



Time taken to complete the task for Splay Tree is minimum than Basic BST and RB-Tree. The reason for this is items that are search here are recently added and those items are much closer to the root when compared to other two Trees. Basic BST has higher time with respect to the RB-Tree. This is because Basic BST could be unbalanced and it might get much time to travel through the nodes when compared to RB-Tree. Further time to complete the task for Basic BST increases while time for RB-Tree decreases with the size of the tree increases also because of the unbalanceness of the Basic BST and balanceness of RB-Tree.

Time taken to **Search** (in Microseconds)					Data set size
Set 2	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	7064	26	20	10
	data_2	16208	71	29	10
	data_3	29642	86	35	10

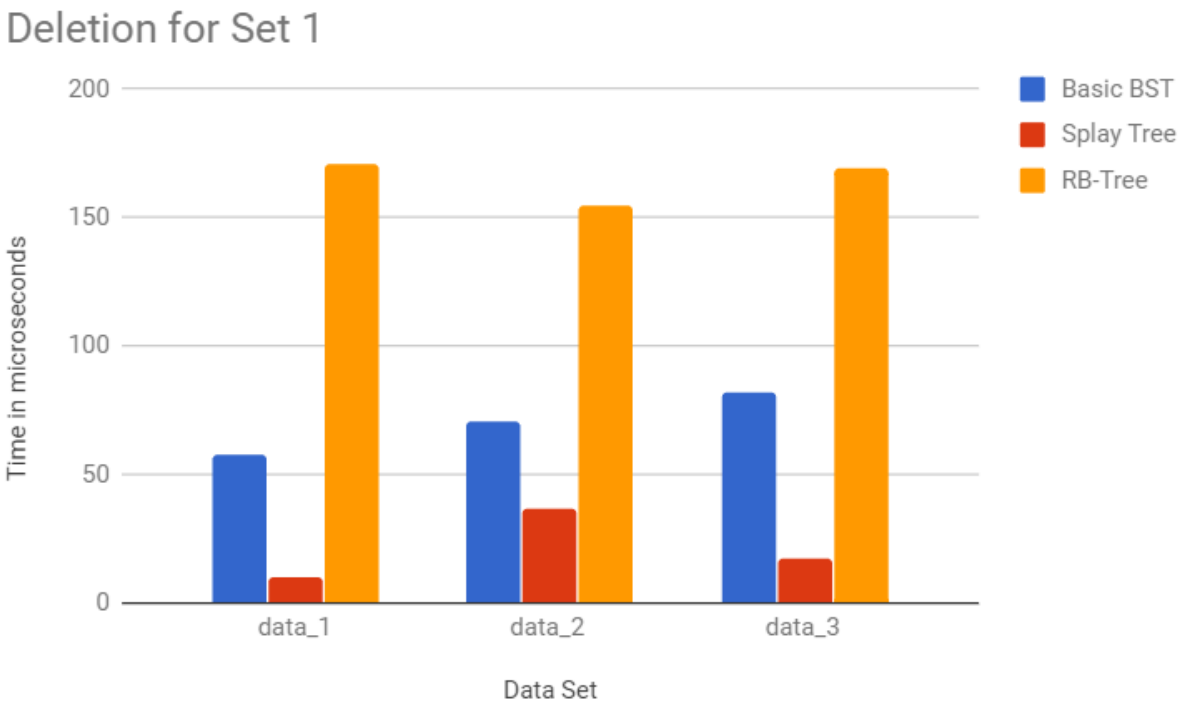


Since the Basic BST is unbalanced for Set 2 as discussed above it takes much time to complete the task compared to other two Trees. As Set two is sorted the Splay Tree also unbalance after insertion. But with more search operations done to the Splay Tree it might balance. But RB-Tree will always be balanced tree. That is the reason to have superior time to Splay Tree than RB-Tree.

c)

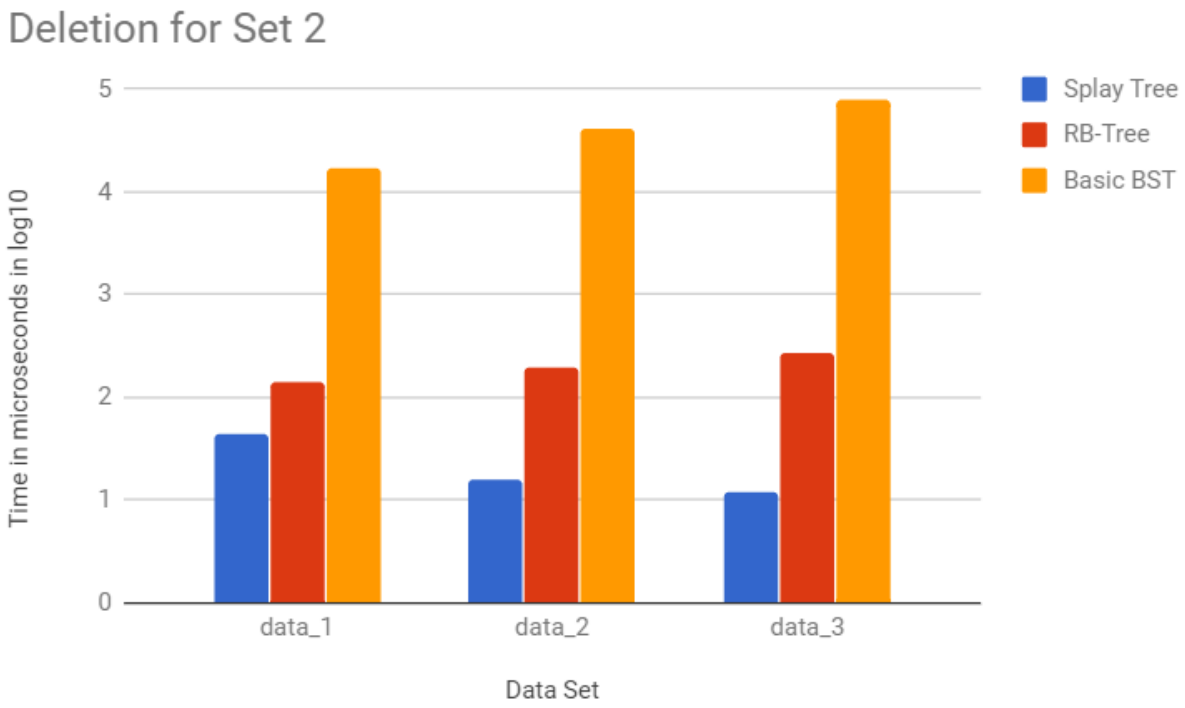
Time taken to **Delete** (in Microseconds)					Data set size
Set 1	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	58	10	171	10
	data_2	71	37	155	10
	data_3	82	17	169	10

Following graph shows the time taken for Set 1 to complete deletion of 10 items in Basic BST, Splay Tree and RB-Tree. Specialty of these 10 items is they are the last 10 items in the insert datasets. Hence, these items are recently added items to each Tree.



Since recently added items are deleted Splay Tree has performed well in deletion as recently added items are close to the root and it takes less time to travel through nodes to delete. Further, RB-Tree has the maximum time because it performs rotation to keep RB-property.

Time taken to **Delete** (in Microseconds)					Data set size
Set 2	Data Set	Basic BST	Splay Tree	RB-Tree	
	data_1	16733	44	141	10
	data_2	42074	16	194	10
	data_3	78712	12	277	10



Here also Basic BST has the maximum time because tree is unbalance as the inserted dataset was sorted. Splay Tree has the minimum time since the deletion is done to the recently added items.

References

1. Introduction to Algorithms (3rd Edition) by Thomas H. Cormen