ASSIGNMENT - 3

Data Structures Laboratory

TREAPS

Node Structure:

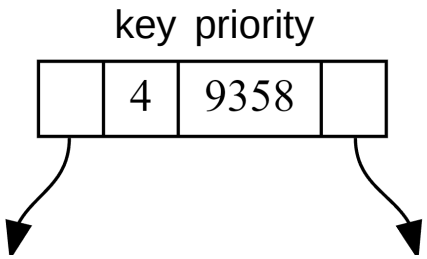
```
class Treap_node
{
      int key;
      int priority;
      Treap_node *left;
      Treap_node *right;
public:
      Treap_node()
      {
      }
      Treap_node(int x)
      {
         key=x;
         priority=rand()%100000;
         left=right=NULL;
      }
     ~Treap_node()
      {
          delete left;
          delete right;
       }
friend class Treap;
};
```

int key represents the key to be inserted into treap.

int priority represents the priority of the inserted element.

left,right points to left and right subtree of a node.

Sample node is shown below.



Treap_Insert(int) :

Insert function gets integer elements as input and generates some random priority for that input and inserts it into treap.

The key element is inserted as in case of how insertion is done in Binary Search Tree.

If the key < parent->key then key is inserted at left of parent by traversing recursively.

If the key> parent-> key then key is inserted at right of parent by traversing recursively.

During creation of node in class Treap_node a random priority is assigned for each node using rand() function.

This random priority helps to maintain heap property .

In my implementation I have implemented the property of MAX Heap.

Once the key got inserted using BST property, we need to check whether the priority follows heap property or not.

During insertion, we recursively traverse all ancestors of the inserted node.

a) If new node is inserted in left subtree and root of left subtree has higher priority, perform right rotation.

b) If new node is inserted in right subtree and root of right subtree has higher priority, perform left rotation.

Treap_Delete(int):

To delete an element from treap we have 3 cases.

Case 1: If the element to be deleted is a leaf just delete it.

Case 2: If the node to deleted has only one child then replace the node with non null child.

Case 3: If the node to be deleted has two children find the maximum of left and right children.

i) If priority of right child is greater, perform left rotation at node

ii) If priority of left child is greater, perform right rotation at node.

We are breaking down the Case 3 to Case 1 or Case 2.

Treap_RotateLeft(Treap_node*)

Create 2 nodes R,X pointing to root ->right and root->right->left

Perform left rotation by changing pointers.

```
R->left = root;
root->right = X;
root = R;
```

Treap_RotateRight(Treap_node*)

Create 2 nodes L,Y pointing to root ->left and root->left->right

Perform left rotation by changing pointers.

```
L->right = root;
root->left = Y;
root = L;
```

Treap_Search(int)

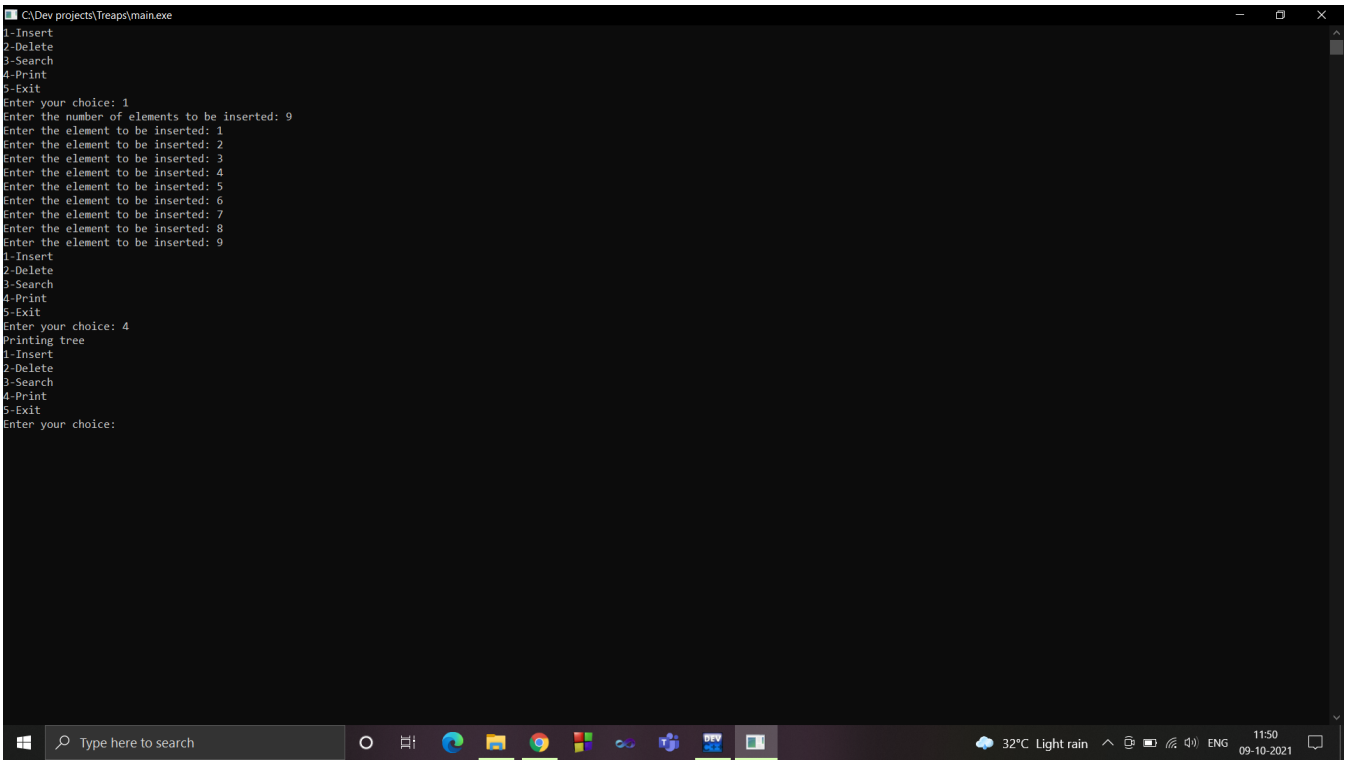
Searching is same as Binary Search Tree.

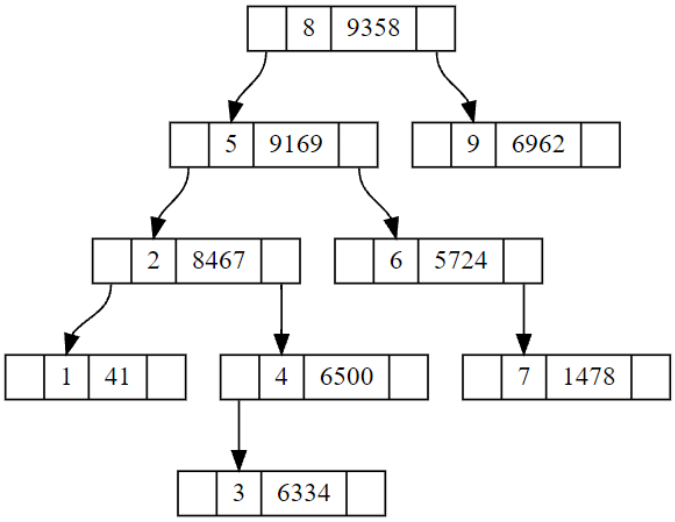
If key < root->key traverse left subtreee.

If key > root->key traverse right subtree.

Sample Test Cases:

INSERT :1,2,3,4,5,6,7,8,9.

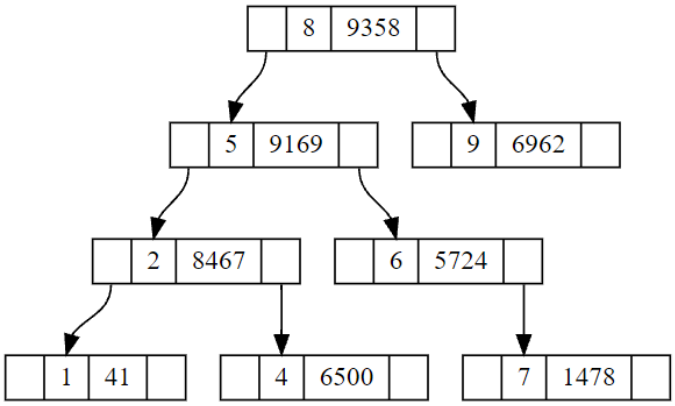




Let the above tree be original tree and further operations are performed on the same tree

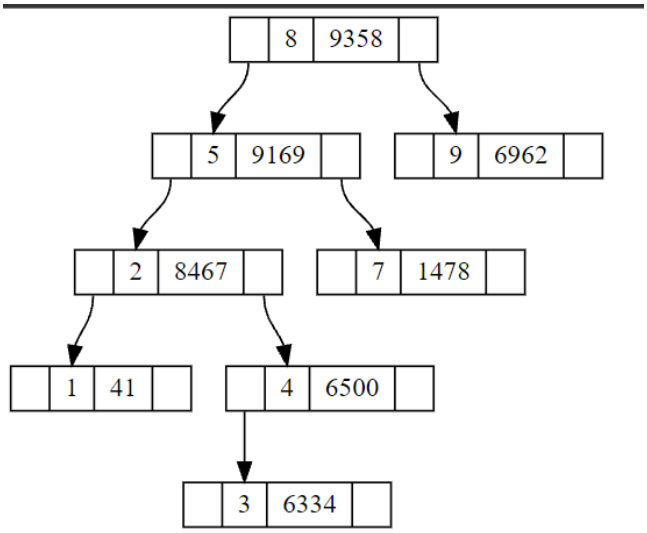
Deleting a leaf.

Delete 3.



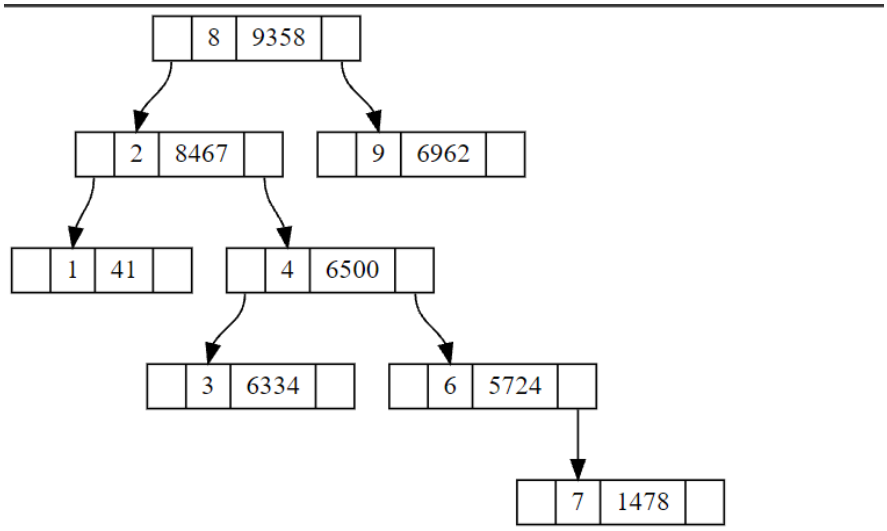
Deleting a node with single child.

Delete 6.



Deleting a node with 2 children.

Delete 5.



Search 3 before and after Deletion.

```
1-Insert
2-Delete
3-Search
4-Print
5-Exit
Enter your choice: 1
Enter the number of elements to be inserted: 9
Enter the element to be inserted: 1
Enter the element to be inserted: 2
Enter the element to be inserted: 3
Enter the element to be inserted: 4
Enter the element to be inserted: 5
Enter the element to be inserted: 6
Enter the element to be inserted: 7
Enter the element to be inserted: 8
Enter the element to be inserted: 9
1-Insert
2-Delete
3-Search
4-Print
5-Exit
Enter your choice: 3
Enter the element to be searched: 3
Element found
1-Insert
2-Delete
3-Search
4-Print
5-Exit
Enter your choice: 2
Enter the element to be deleted: 3
1-Insert
2-Delete
3-Search
4-Print
5-Exit
Enter your choice: 3
Enter the element to be searched: 3
Element not found
1-Insert
2-Delete
3-Search
4-Print
5-Exit
```

Parameter Evaluation
(BST vs AVL vs Treap)

Parameteres choosen:

i) Number of comparisions
ii) Number of rotations
iii) Average Height of each node
iv) Final Height of the Tree

i) Number of Comparisions

I have created 3 variables for 3 Data Structures

Each of them will be incremented when comparision is done.

For file containing 250 operations the number of comparisions are displayed as output.

BST Comparisions: 2010

AVL Comparisions: 1836

AVL Comparisions including Balance Factor comparisions: 2841

Treap Comparisions: 1671

Treap Comparisions including Priority Comparisions: 2889

For file containing 500 operations

BST Comparisions: 3989

AVL Comparisions: 4025

AVL Comparisions including Balance Factor comparisions: 6366

Treap Comparisions: 3607

Treap Comparisions including Priority Comparisions: 6213

For file containing 1500 operations

BST Comparisions: 14323

AVL Comparisions: 14177

AVL Comparisions including Balance Factor comparisions: 21728

Treap Comparisions: 14475

Treap Comparisions including Priority Comparisions: 24335

For file containing 6000 operations.

BST Comparisions: 79614

AVL Comparisions: 68174

AVL Comparisions including Balance Factor comparisions: 102224

Treap Comparisions: 73579

Treap Comparisions including Priority Comparisions: 124309

For file containing 30000 operations.

BST Comparisions: 493391

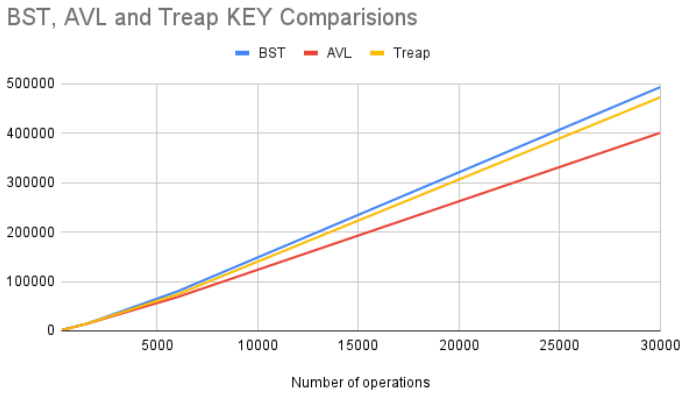
AVL Comparisions: 400886

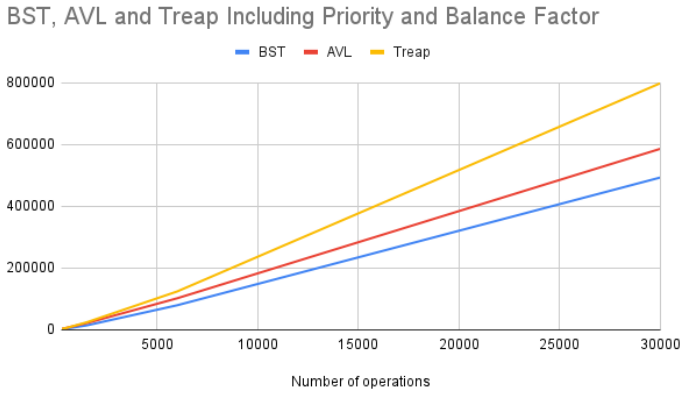
AVL Comparisions including Balance Factor comparisions: 586348

Treap Comparisions: 472761

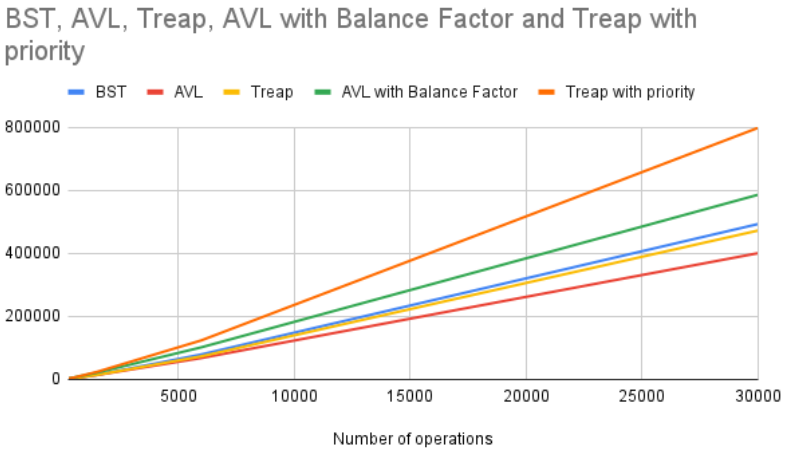
Treap Comparisions including Priority Comparisions: 798729

The overall comparision is shown by the below graphs.





The final Graph displaying comparisions of all 3 Data structures.



This parameter number of comparisions is choosen as it is being done in all the 3 data structures while inserting and deleting an element.

For each element to be inserted or deleted comparision needs to done, so it is an important parameter

ii) Number of Rotations

Rotations is done only in AVL and Treaps.

I have created 2 variables for 2 Data Structures.

Each of them will be incremented when rotation is performed.

Double Rotation is considered as 2 single rotations.

For file containing 250 operations

Rotations

AVL Rotations: 134

Treap Rotations: 339

For file containing 500 operations

AVL Rotations: 317

Treap Rotations: 722

For file containing 1500 operations

AVL Rotations: 883

Treap Rotations: 2414

For file containing 6000 operations

AVL Rotations: 3432

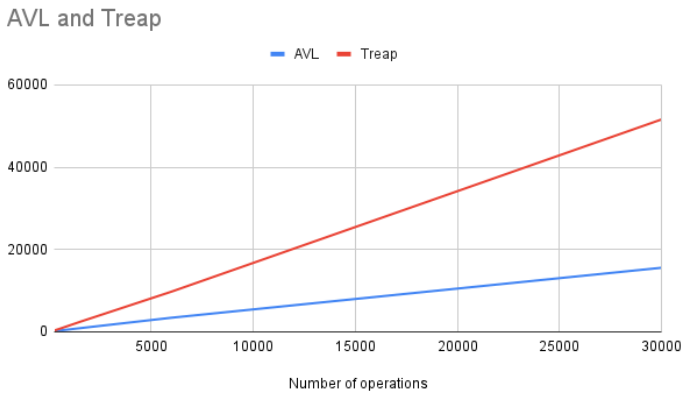
Treap Rotations: 9774

For file containing 30000 operations

AVL Rotations: 15561

Treap Rotations: 51575

Graph representation:



Number of rotations is choosen as a parameter as it balances the tree by reducing its height in some cases.

iii) Average Height of each node

I have created two functions.

One function to count number of nodes and another function to find sum of heights of all the nodes.

By dividing the value of sum of heights of all nodes with number of nodes ,we get the average height of each node.

For file containing 250 operations

Average height of Nodes

BST: 3.21

AVL: 2.25253

Treap: 3.18182

For file containing 500 operations

Average height of Nodes

BST: 3.26562

AVL: 2.25131

Treap: 3.21875

For file containing 1500 operations

Average height of Nodes

BST: 3.20479

AVL: 2.2967

Treap: 3.41739

For file containing 6000 operations

Average height of Nodes

BST: 3.14879

AVL: 2.30777

Treap: 3.42539

For file containing 30000 operations

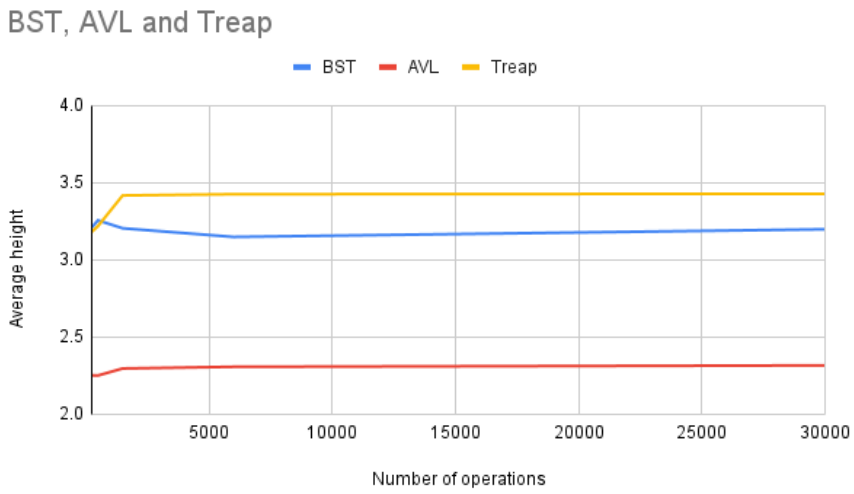
Average height of Nodes

BST: 3.19825

AVL: 2.31671

Treap: 3.42824

Graphical representation is shown below.



Average height of the tree is choosen as a parameter as it shows how balanced the tree is.

Average height of each node is less in AVL tree as it is height balanced tree.

iv) Final Height of the Tree

I have created a function to Calculate the height of the tree by passing root as the argument

Height of the tree is calculated by

$$H = 1 + \max\{\text{height(left subtree)}, \text{height(right subtree)}\}$$

For file containing 250 operations

Height of the tree

BST Height: 12

AVL Height: 8

Treap Height: 13

For file containing 500 operations

Height of the tree

BST Height: 14

AVL Height: 9

Treap Height: 17

For file containing 1500 operations

Height of the tree

BST Height: 15

AVL Height: 10

Treap Height: 19

For file containing 6000 operations

Height of the tree

BST Height: 20

AVL Height: 13

Treap Height: 22

For file containing 30000 operations

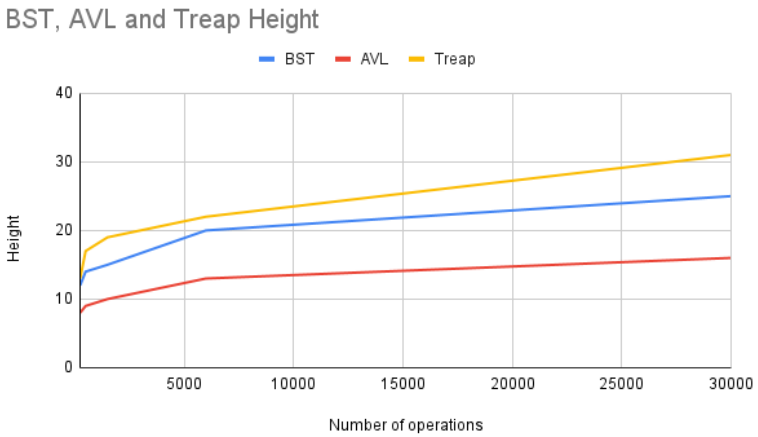
Height of the tree

BST Height: 25

AVL Height: 16

Treap Height: 31

Graph representation is shown below.



Final height of the tree is choosen as a parameter for evaluation as it affects the search time of the tree.

Search time of element can go upto O(log H)

where H is the height of the tree.