**Comparison Report**

*214101019*

**Test Case Generation :**

* For test case generation rand() function is used
* The test case generator generates 5 testcase files
* The ratio of insert to delete is 70:30
* For 5 test cases the input contains 10K,12K,14K,16K,18K operations ie. Insert and delete.
* The code for test case generation is as under:
* void testcase\_generator()

{

*int i = 10000,k = -1,testcases=1;*

*char s[100] = "testcase\_";*

*for(testcases=1;testcases<=5;testcases++)*

*{*

*char name[60] = "";*

*const char \*filename= {0};*

*sprintf(name,"%s%d.txt",s,testcases);*

*filename = name;*

*FILE \* fptr = fopen(filename,"w");*

*set <int> used\_key;*

*int operations = i +2000\*(testcases-1);*

*int o ,key ;*

*fprintf(fptr,"%d\n",operations);*

*while (operations > 0 )*

*{*

*o = rand()%2;*

*key = rand()%30000;*

*int p =70,q=30;*

*while(p && operations>0)*

*{*

*key = rand()%30000;*

*fprintf(fptr , "Insert %d\n",key);*

*used\_key.insert(key);*

*p--;*

*operations--;*

*}*

*while(q && operations>0)*

*{*

*key = rand()%30000;*

*fprintf(fptr , "Delete %d\n",key);*

*used\_key.erase(key);*

*q--;*

*operations--;*

*}*

*}*

*fclose(fptr);*

*}*

*}*

Tabular Representation:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Testcase1 | Testcase2 | Testcase3 | Testcase4 | Testcase5 |
| Total operations | 10000 | 12000 | 14000 | 16000 | 18000 |
| Insert | 7000 | 84000 | 98000 | 11200 | 12600 |
| Deletion | 3000 | 36000 | 42000 | 4800 | 5400 |

* The test samples generated are random

**Comparison Report Generation:**

* For comparison of Treap, AVL tree , BST following parameters are used:

1. HEIGHT OF THE TREE
2. #COMPARISONS
3. #ROTATIONS
4. AVERAGE HEIGHT OF THE TREE

Note:

In all the sebequent graphs Blue Color represents TREAP

Orange color represents AVL tree

Gray color represents BST

Discussing about the parameters :

1. **HEIGHT OF THE TREE:**

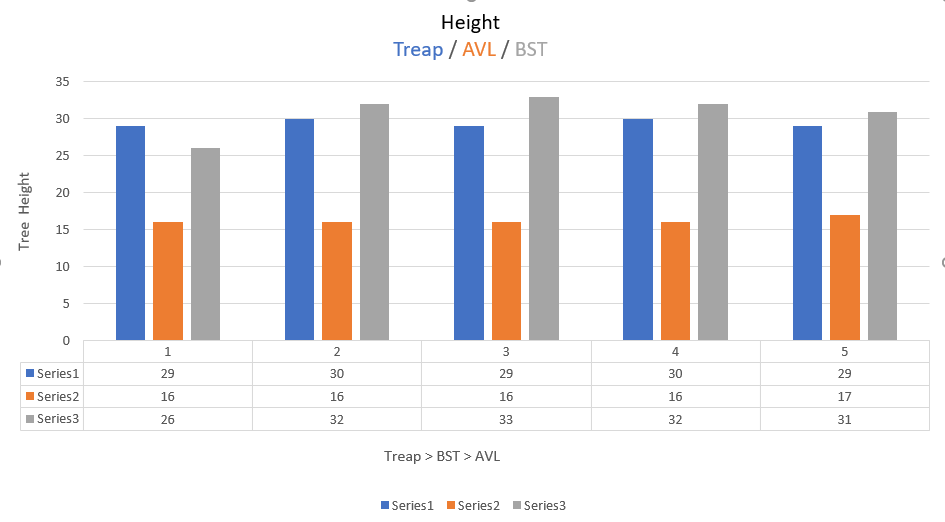
* AVL tree is a self balancing tree
* Using the idea of balance factor is maintains a balanced height
* Whenever a node is inserted or deleted into and from the AVL tree respectively then
* If the tree becomes unbalanced then the avl tree balcances it self by performing rotations
* However this is not the case in treap and bst
* Incaase of BST no balancing is performed at all
* Incase of Treap balancing is performed but only to maintain heap property.

**Theoretically:**

* Because AVL is a self balancing tree thus the height of the tree is O(logn) to O(1.44logn)
* The height of treap ranges form O(logn) to O(n)
* The height of bst ranges from O(logn) to O(n)

**Practically:**

* It is evident from the 5 testcases report that
  + the height obtained for AVL is the minimum
  + the height of BST and TREAP is almost similar
  + However, height of bST is slightly more than that of treap in most case.
* **Graphically:**



1. **#COMPARISONs:**

* For comparisons whenever a node is compared with another node.
* Then the number of comparisons is incremented by one.
* Incase of TREAP:
  + Whenever a node is compared with another node in terms of key value or in terms of priority
  + Then the value of number of comparisons is incrementd by 1.
* Incase of BST:
  + Whenever a node is compared with another node in terms of key value
  + Then the value of number of comparisons is incrementd by 1.
* Incase of BST:
  + Whenever a node is compared with another node in terms of key value
  + Then the value of number of comparisons is incrementd by 1.
* The value incrementation takes place when we traverse the tree whenever the node is inserted or deleted from the tree.

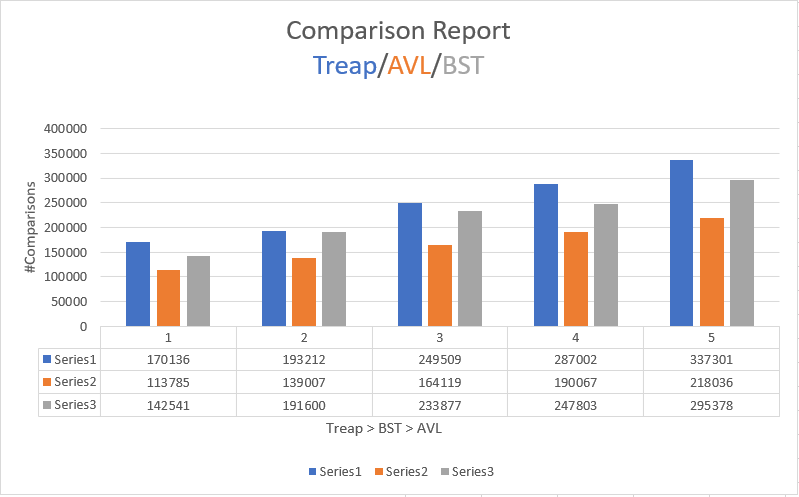
**Theoretically:**

* AVL is self balancing tree so the height or depth is proportion to O(logn)
* In case of TREAP and BST the height or the depth varies from O(longn) to O(n).
* We know the while inserting or deleting we may need to traverse the height of the tree and thus the number of comparisons comes out to be proportional to the height of the tree.
* Thus the number of comparisons is minimum in case of AVL and about similar in case of TREAP and BST.

**Practically:**

* It is evident from the 5 testcases report that
  + the #comparisons obtained for AVL is the minimum
  + the #comparisons of BST and TREAP is almost similar
  + However #comparisons of treap is slightly more than that of bst in every case possibly because it compares between priority and key value both.

Graphically Comparing:



1. **#ROTATIONS**

* AVL tree is a self balancing tree
* Using the idea of balance factor is maintains a balanced height
* Whenever a node is inserted or deleted into and from the AVL tree respectively then
* If the tree becomes unbalanced then the avl tree balcances it self by performing rotations
* Whenever insertion of a node takes place the number fo rotations can go upto max 2 rotations and min 0 that is in case of no rotations.
* Whenever a node is deleted then the number of rotations can go upto minimum 0 and maximum can go upto the height of the tree.
* AVL comprises of single as well as double rotations.
* However this is not the case in treap and bst
* Incase of BST no balancing is performed at all
* So the number of rotations in BST is 0.
* Incase of Treap balancing is performed but only to maintain heap property and it comprises of only single rotations.
* However whenever a node is inserted then the rotations can go upto the height of the tree
* Similarly whenever a node is deleted then the rotations can go upto the height of the tree i.e. from root to the leaf node.

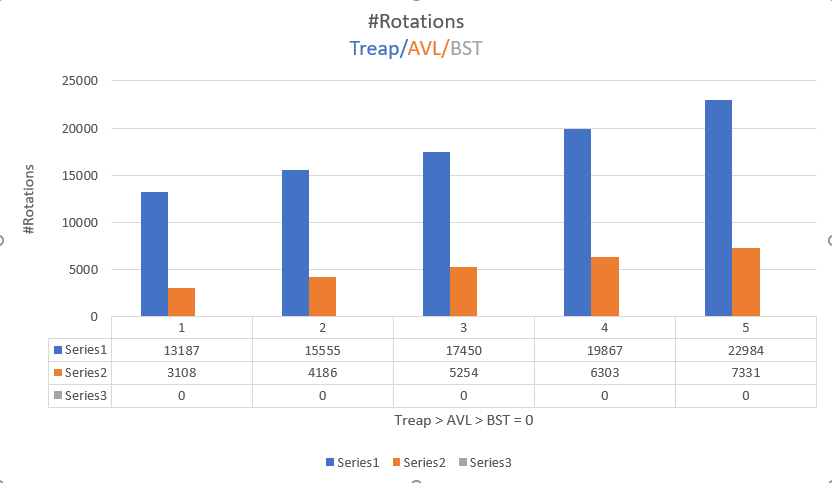
**Theoretically:**

* Incase of Treap number of rotations is maximum as both during inertion and deletion the max rotations can be of order of height
* Incase of AVL the number of rotations is less than in Treap and constant number of rotations in insertion and O(n) rotations is case of deletion.
* Incase of BST no rotaions are performed and it is 0.

**Practically:**

* It is evident from the 5 testcases report that
  + #rotations in case of treap is maximum in every case
  + the #comparisons of AVL is less than that of Treap
  + the #rotations obtained for BST is the minimum = 0

Graphically:



1. **Average height**

* The average height is proportional to the height of the tree
* Incase of AVL which is self-balancing
* Each node is balanced and thus the as each node is balanced the height of the node is also the minimum one.
* In case of Treap rotations are performed and thus the height of the node reduces a little
* Incase of BST no rotations are performed and thus each node is hight greater than or equal to that in cse of treap and avl tree.

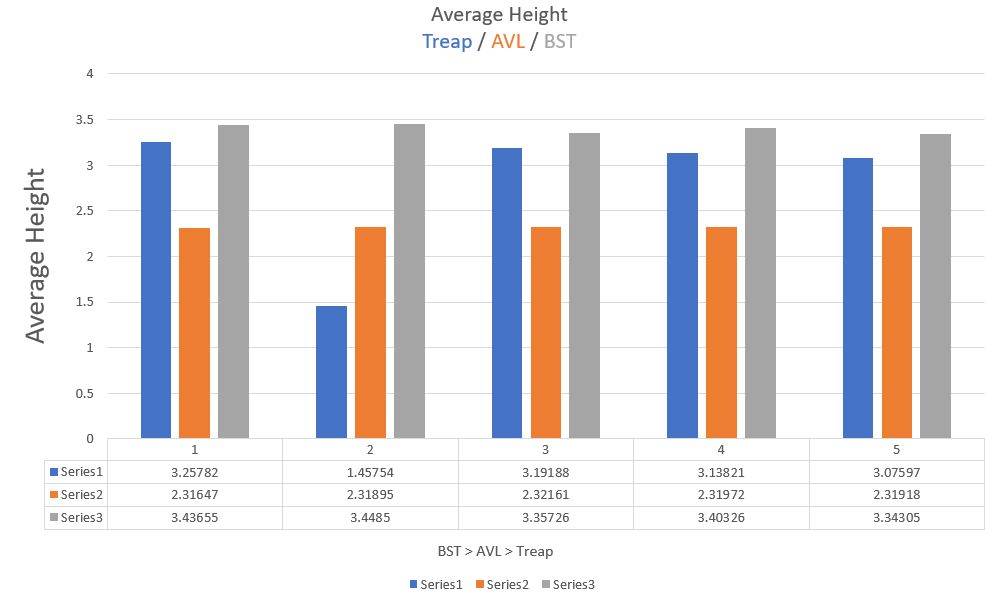
**Theoretically:**

* Average height of the bst is the maximum
* Average height of the treap is less than that of bst but greater than that of bst
* Average height of the avl is minimum as the avl tree is self-balancing tree.

**Practically:**

* It is evident from the 5 testcases report that
  + Average hight of node in case of BST is maximum in every case
  + the average height of Treap is less than that of BST but greater than that of AVL in most cases.
  + the average height of the AVL tree is the minimum

**Graphically Comparing:**



* **Height of the tree calculation function:**

The below funciton calls node\_height function on root which results in calculation of the height of the tree

findHeight()

{

int h = -1;

tree\_height = node\_height(head->rchild);

avg\_height = avg\_height/total\_nodes;

}

* **node\_Height calculation function:**

This function calculates the height of each node by recursively calling the node\_height function on its left and right subtree

node\_height(Treap\_Node\* temp)

{

if (temp == NULL)

return 0;

int subtree\_right = node\_height(temp->rchild);

int subtree\_left = node\_height(temp->lchild);

int hnode = (subtree\_right>subtree\_left)?subtree\_right+1:subtree\_left+1;

avg\_height += hnode;

return hnode;

}