**Topic – Operations in Binary Search Tree , Insertion , Deletion , Searching , Traversal,Printing**

* **Problem Statement**

Write Program to do the following:  
  
1) Insert n integers in BST, where n integers are generated at random. With different values of n, ie, 50, 100, 1000, 10000, 25000, 50000 etc. create the BST. For each n value, run your program for 50 different permutations of random numbers. For each n value compute the total height  to insert the n elements in each of the 50 cases and report the average height. Verify if it follows O(n log n).  
  
2) Search for an element.  
  
3) Delete an element ( consider all possible cases)  
  
4) Print the BST in inorder traversal (Iterative)

**Input and Output example:**

The average height of the binary search tree for 50 elements is 10.860

The average height of the binary search tree for 100 elements is 14.560

The average height of the binary search tree for 1000 elements is 22.520

The average height of the binary search tree for 10000 elements is 31.880

The average height of the binary search tree for 25000 elements is 35.060

The average height of the binary search tree for 50000 elements is 37.900

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

56

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

78

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

90

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

24

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The inorder traversal of present binary seach tree is : 23 24 56 78 90

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

3

Enter the element to be deleted

23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The inorder traversal of present binary seach tree is : 24 56 78 90

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the element to be inserted

23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

3

Enter the element to be deleted

23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The inorder traversal of present binary seach tree is : 24 56 78 90

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

2

Enter the element to be searched

56

56 is present in the binary search tree

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

3

Enter the element to be deleted

56

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The inorder traversal of present binary seach tree is : 24 78 90

\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0:Exit

1:Insert an element in the binary search tree

2:Search for an element in the binary search tree

3:Delete an element

4:Print the binary search tree in inorder traversal

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

0

* **Proposed C Code**

**/\* ---------- bst\_operation.c--------------- \*/**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**#define SIZE 100000**

**typedef struct node**

**{**

**int data;**

**struct node\* lchild;**

**struct node\* rchild;**

**}Tree;**

**static int top = -1;**

**void stackempty()**

**{**

**printf("Stack is empty\n");**

**return;**

**}**

**void stackfull()**

**{**

**printf("Stack is full\n");**

**exit(1);**

**}**

**/\* function to push item \*/**

**void push(Tree\*\* stack,Tree\* root)**

**{**

**if(top>=SIZE-1)**

**stackfull();**

**else**

**stack[++top] = root;**

**}**

**/\* function to pop \*/**

**Tree\* pop(Tree\*\* stack)**

**{**

**if(top==-1)**

**stackempty();**

**else**

**return stack[top--];**

**}**

**/\* creating a node for bst \*/**

**Tree\* creation(int data)**

**{**

**Tree\* ptr = (Tree\*)malloc(sizeof(Tree));**

**ptr->data = data;**

**ptr->lchild = NULL;**

**ptr->rchild = NULL;**

**return ptr;**

**}**

**/\* inserting in bst \*/**

**Tree\* insertion(Tree\* root,int data)**

**{**

**Tree\* temp = creation(data);**

**Tree\* check = NULL;**

**Tree\* ptr = root;**

**while ( ptr != NULL )**

**{**

**check = ptr;**

**if(( temp->data ) < ( ptr->data ))**

**{**

**ptr = ptr->lchild;**

**}**

**else**

**{**

**ptr = ptr->rchild;**

**}**

**}**

**if ( check == NULL )**

**{**

**root = temp;**

**}**

**else**

**{**

**if((temp->data) < (check->data))**

**{**

**check->lchild = temp;**

**}**

**else**

**{**

**check->rchild = temp;**

**}**

**}**

**return root;**

**}**

**/\* calculating height of a bst \*/**

**int bst\_height(Tree\* root)**

**{**

**if ( root == NULL )**

**{**

**return 0;**

**}**

**else**

**{**

**int left\_height = bst\_height(root->lchild);/\* height in the left portion of subtree \*/**

**int right\_height = bst\_height(root->rchild);/\* height in the right portion of subtree \*/**

**if ( left\_height > right\_height )**

**{**

**return left\_height + 1;**

**}**

**else**

**{**

**return right\_height + 1;**

**}**

**}**

**}**

**/\* permutation of the random numbers \*/**

**void permutation( int\* arr,int n)**

**{**

**srand(time(0));**

**int random\_index1,random\_index2;**

**for ( int i = 0,j = n-1 ; i < j ; i++,j-- )**

**{**

**/\* swaping randomly \*/**

**random\_index1 = rand()%(i+1);**

**random\_index2 = rand()%(j+1);**

**int temp = arr[random\_index1];**

**arr[random\_index1] = arr[random\_index2];**

**arr[random\_index2] = temp;**

**}**

**}**

**/\* inorder traversal to print the nodes of bst \*/**

**void inorderTraversal(Tree\* root)**

**{**

**if(root == NULL)**

**{**

**printf("No node is present in binary search tree\n");**

**return;**

**}**

**printf("The inorder traversal of present binary seach tree is : ");**

**Tree\*\* stack = (Tree\*\*)malloc(SIZE\*sizeof(Tree\*));**

**Tree\* temp = root;**

**while (( temp != NULL ) || ( top != -1 ))**

**{**

**while ( temp != NULL )**

**{**

**push(stack,temp);**

**temp = temp->lchild;**

**}**

**temp = pop(stack);**

**printf("%d ",temp->data);**

**temp = temp->rchild;**

**}**

**printf("\n");**

**free(stack);**

**}**

**/\* searching an element \*/**

**Tree\* search\_element(Tree\* root,int element)**

**{**

**if ( root == NULL )**

**{**

**printf("%d is not present\n",element);**

**return NULL;**

**}**

**if ( root->data == element )**

**{**

**printf("%d is present in the binary search tree\n",element);**

**return root;**

**}**

**if ( element < root->data )**

**{**

**return search\_element(root->lchild,element);**

**}**

**else**

**{**

**return search\_element(root->rchild,element);**

**}**

**}**

**/\* finding minimum element \*/**

**Tree\* min\_element(Tree\* root)**

**{**

**Tree\* ptr = root;**

**while (( ptr!= NULL ) && ( ptr->lchild != NULL ))**

**{**

**ptr = ptr->lchild;**

**}**

**return ptr;**

**}**

**/\* deleting a node \*/**

**Tree\* delete\_element(Tree\* root,int element)**

**{**

**if ( root == NULL)**

**{**

**printf("Element to be deleted is not present in the binary search tree\n");**

**return root;**

**}**

**if ( element < root->data )**

**{**

**root->lchild = delete\_element(root->lchild,element);**

**}**

**else if ( element > root->data )**

**{**

**root->rchild = delete\_element(root->rchild,element);**

**}**

**else**

**{**

**if ( root->lchild == NULL )/\* if left child is absent \*/**

**{**

**return root->rchild;**

**}**

**else if ( root->rchild == NULL )/\* if right child is absent \*/**

**{**

**return root->lchild;**

**}**

**/\* The case when root has both children \*/**

**Tree\* temp = min\_element(root->rchild);**

**root->data = temp->data;**

**root->rchild = delete\_element(root->rchild,temp->data);**

**}**

**return root;**

**}**

**int main()**

**{**

**Tree\* root = NULL;/\* initially root is NULL \*/**

**int treesize[] = {50,100,1000,10000,25000,50000};**

**srand(time(0));**

**for ( int i = 0 ; i < 6 ; i++ )**

**{**

**int\* arr = (int\*)malloc(treesize[i]\*sizeof(int));**

**for ( int j = 0 ; j < treesize[i] ; j++ )**

**{**

**arr[j] = rand();**

**}**

**int height = 0;**

**/\* generating permutation for 50 numbers \*/**

**for ( int j = 0 ; j < 50 ; j++ )**

**{**

**permutation(arr,treesize[i]);**

**root = NULL;**

**for ( int k = 0 ; k < treesize[i] ; k++ )**

**{**

**root = insertion(root,arr[k]);**

**}**

**height += bst\_height(root);/\* total height for 50 cases \*/**

**}**

**/\* dividing total height by 50 \*/**

**printf("The average height of the binary search tree for %d elements is %0.3f\n",treesize[i],(float)height/50);**

**free(arr);**

**}**

**root = NULL;**

**int option,element;**

**do{**

**/\* the options are given \*/**

**printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*List of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("0:Exit\n");**

**printf("1:Insert an element in the binary search tree\n");**

**printf("2:Search for an element in the binary search tree\n");**

**printf("3:Delete an element\n");**

**printf("4:Print the binary search tree in inorder traversal\n");**

**printf("\n\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*\n");**

**scanf("%d",&option);**

**switch(option)/\* the cases are designed according to the opertions listed above \*/**

**{**

**case 1:**

**printf("Enter the element to be inserted\n");**

**scanf("%d",&element);**

**root = insertion(root,element);**

**break;**

**case 2:**

**printf("Enter the element to be searched\n");**

**scanf("%d",&element);**

**root = search\_element(root,element);**

**break;**

**case 3:**

**printf("Enter the element to be deleted\n");**

**scanf("%d",&element);**

**root = delete\_element(root,element);**

**break;**

**case 4:**

**inorderTraversal(root);**

**break;**

**case 0:**

**break;**

**default:**

**printf("Please give option withen (0-4)\n");**

**}**

**}while(option!=0);/\* when option is zero then it will exit \*/**

**return 0;**

**}**

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

* **Conclusion**

**The proposed algorithm has average time complexity of O(nlogn) where n is the input size i.e the number of elements.**

* **Limitations : As stack size is taken 100000 so inoder traveral for n>= 100000 will be difficult .**
* **Assumptions: User should give integer values in the binary search tree.**