K L UNIVERSITY

FRESHMAN ENGINEERING DEPARTMENT

A Project Based Lab Report

On

BINARY SEARCH TREE AND ITS OPEARTION

SUBMITTED BY:

I.D NUMBER	NO
2200010122	Rishendra
2200010127	lasya Nayanika
2200010129	Vishnu vardhan Reddy
2200010133	Indu Priya
2200010137	Anirudh

UNDER THE ESTEEMED GUIDANCE OF

DR.NAVEEN KUMAR

ASSITANT PROFESSOR

KL UNIVERSITY

Green fields, Vaddeswaram – 522 502 Guntur Dt., AP, India.

DEPARTMENT OF BASIC ENGINEERING SCIENCES

CERTIFICATE

This is to certify that the project based laboratory report entitled "**BINARY SEARCH TREE AND ITS OPERATIONS**" submitted by **Rishendra, Lasya nayanika, Vishnu vardhan, Indupriya , Anirudh** bearing Regd. No. 2200010122, 2200010127, 2200010129, 2200010133, 2200010137, to the **Department of Basic Engineering Sciences, KL University** in partial fulfillment of the requirements for the completion of a project in "**Data Structures (19SC1202)** " course in I B Tech II Semester, is a bonafide record of the work carried out by them under my supervision during the academic year 2019-20.

PROJECT SUPERVISOR	HEAD OF THE DEPARTMENT
Dr.NAVEEN KUMAR	Dr. D.HARITHA

ACKNOWLEDGEMENTS

It is great pleasure for me to express my gratitude to our honorable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project based laboratory report.

I express the sincere gratitude to our director **Dr. A Jagadeesh** for his administration towards our academic growth.

I express sincere gratitude to our Coordinator and HOD-BES **Dr. D.Haritha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor **Mr. Naveen Kumar** for his/her novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Regd . No	Name
2200010122	Rishendra
2200010127,	Lasya
2200010129,	Vishnu vardhan
2200010122,	Indupriya
2200010137	Anirudh

ABSTRACT

In this project we have to implement a binary search tree .Then create a menu driven that allows user to do the insertion, deletion, preorder, inorder, postorder and print the data. This binary search trees supports everything you can get from a sorted array: efficient search, in-order forward/backwards traversal from any given element, search with the added benefit of efficient inserts and deletes.

Any node in a binary tree data structure can be reached by starting at the root node and repeatedly following references to either the left or the right child. The degree, or number of children a node has, is a maximum of two.

This technique is also called as ordered binary tree.In this we will implement all operations in binary search tree .For that we have to insert the values with the insertion operation and then we do inorder,preorder,postorder,searching,delete any element in given tree.But deletion operation is tricky one we have to check many operations .

Modules

- Insertion
- Deletion
- Search
- Preorder
- Inorder
- Postorder

INDEX

INTRODUCTION

Binary Search Tree (BST) is a special binary tree where the key in each node must be greater than or equal to any key stored in the left subtree, and less than or equal to any key stored in the right subtree.

BSTs allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items or lookup tables that allow finding an item by its key .

Search, insert, and delete are the main operations of BST.

As the keys are stored in a particular order, the principle of binary search can be used for lookup.

We start from the root and compare the key. if the root has the key, it is returned as the result.

If the key is greater than the root, the right subtree is recursively checked.

If the key is less than the root, the left subtree is recursively checked.

The recursion continues until the key is found.

The process for visiting all the nodes is called a traversal, and consists of three types: 1) preorder, 2) postorder and 3) inorder.

Insertion and deletion operations are much similar, as both use the same search logic to locate the needed node.

AIM

To implement BST tree and its operations using java

Advantages:-

- Binary Search Tree is fast in insertion and deletion etc. when balanced.
- Very efficient and its code is easier than other data structures.
- Stores keys in the nodes in a way that searching, insertion and deletion can be done efficiently .

Disadvantages:-

- It requires elements to be sorted
- The shape of the binary search tree totally depends on the order of insertions, and it can be degenerated.
- When inserting or searching for an element in binary search tree, the key of each visited node has to be compared with the key of the element to be inserted or found, i.e., it takes a long time to search an element in a binary search tree.

Future enhancements:-

With this we must know that the binary search tree is a different way of structuring data so that it can still be binary searched but it's easier to add and remove elements. The main reason to use a binary search tree is the fact that it extends the capability of a normal array.

SYSTEM REQUIREMENTS

- **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

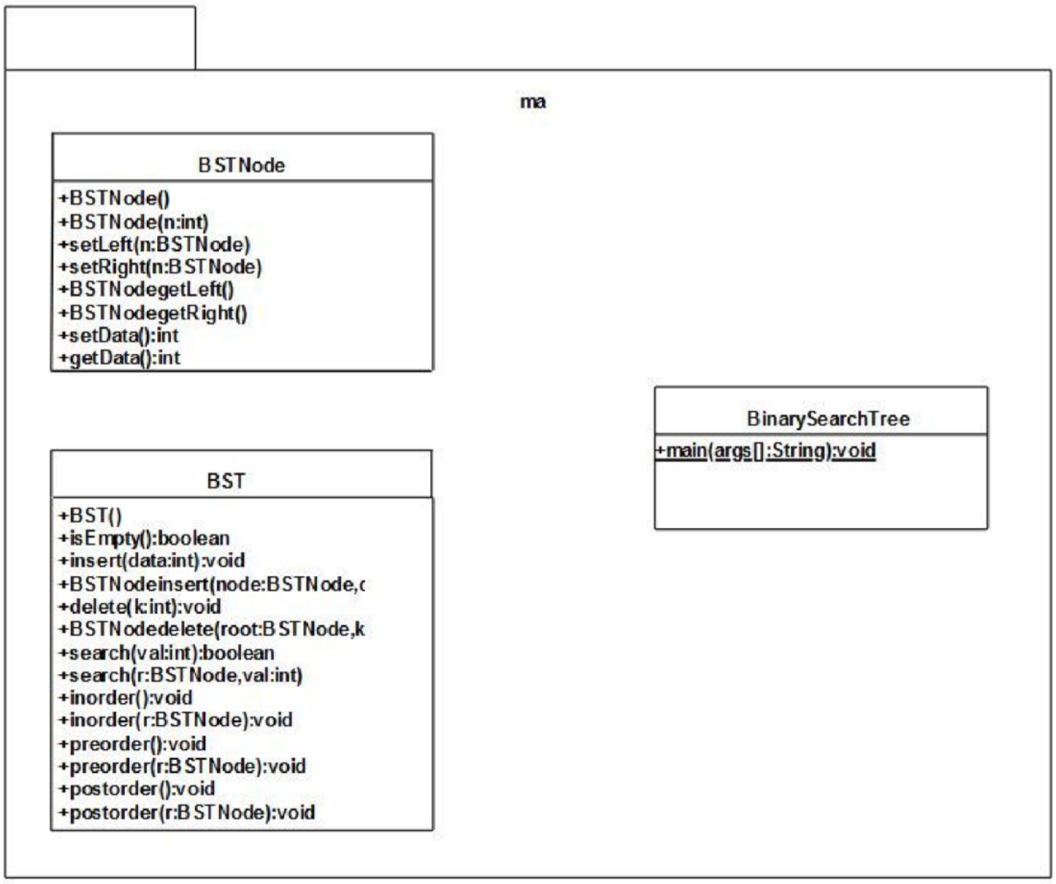
Language	: java:eclipse
Operating system**:**	Windows 10 or later.

- **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM	: 8gb
Processor	: i5

CLASS DIAGRAM



IMPLEMENTATION

```
package ma; public
class BSTNode {
BSTNode left, right; int
data; public BSTNode()
{ left = null;
right = null;
data = 0;
}
public BSTNode(int n) {
left = null; right = null;
data = n;
}
public void setLeft(BSTNode n)
{ left =
n;
}
public void setRight(BSTNode n)
{ right =
n;
}
public BSTNode getLeft()
{ return
left;
}
public BSTNode getRight()
{
return right;
}
public void setData(int d)
{
data = d;
}
public int getData()
{
return data;
}  }
pu
bli
c
cla
```

```
ss
BS
T {
pu
bli
c
BS
TN
ode
roo
t;
public BST()
{ root =
null;
}
public boolean isEmpty()
{
return root == null;
}
public void insert(int data)
{
root = insert(root, data);
}
public BSTNode insert(BSTNode node, int data)
{ if (node == null) node =
new BSTNode(data);
else
{
if (data <= node.getData()) node.left
= insert(node.left, data); else
node.right = insert(node.right, data);
}
return node;
}
public void delete(int k)
{ if
(isEmpty())
System.out.println("Tree Empty"); else
if (search(k) == false)
System.out.println("Sorry "+ k +" is not present");
else {
root = delete(root, k);
System.out.println(k+ " deleted from the tree");
}
}
public BSTNode delete(BSTNode root, int k)
{
BSTNode p, p2, n;
if (root.getData() == k)
{
BSTNode lt, rt; lt
= root.getLeft(); rt
= root.getRight();
if (lt == null &&
rt == null) return
null; else if (lt ==
null)
{ p = rt;
return p;
} else if (rt ==
null)
{ p =
lt;
return p;
} else { p2 = rt; p = rt;
while (p.getLeft() != null)
p = p.getLeft();
p.setLeft(lt); return p2;
}
}
if (k < root.getData())
{
n = delete(root.getLeft(), k);
root.setLeft(n); } else
{
n = delete(root.getRight(), k); root.setRight(n);
}
return root;
}

public boolean search(int val)
{
return search(root, val);
}

public boolean search(BSTNode r, int val)
{
```

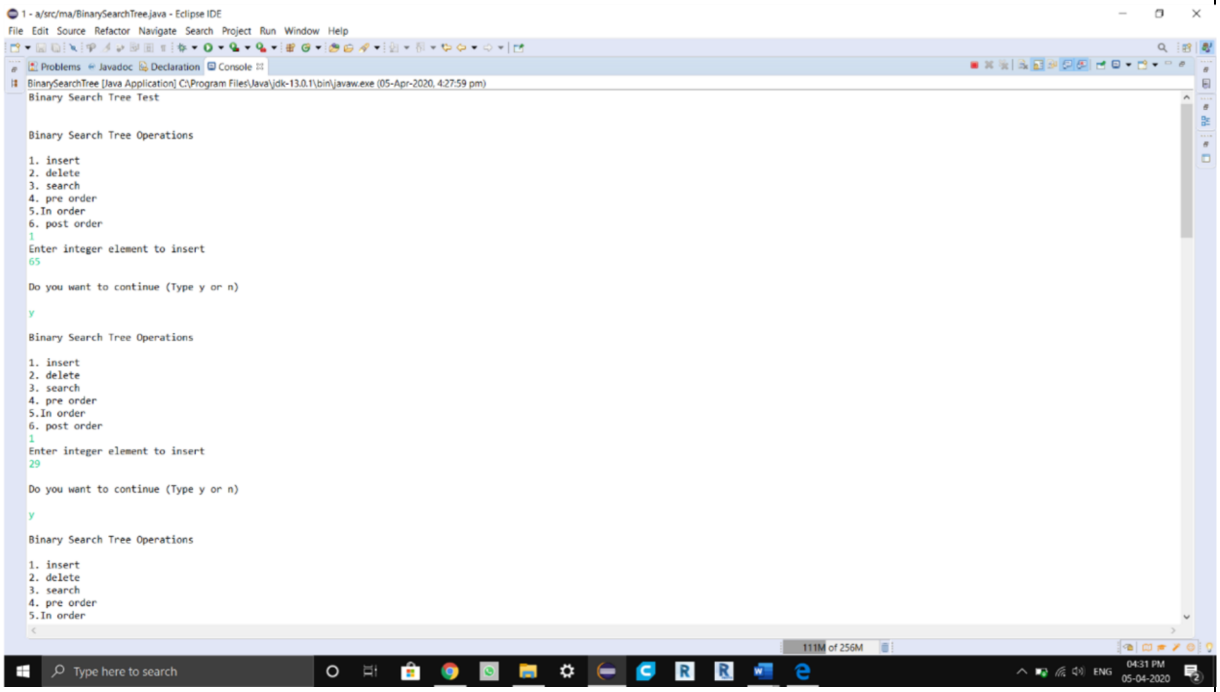
```
boolean found = false;
while ((r != null) && !found)
{
int rval = r.getData();
if (val < rval) r =
r.getLeft(); else if (val
> rval) r =
r.getRight(); else
{
found = true; break;
}
found = search(r, val);
}
return found;
}

public void inorder()
{
inorder(root);
}
public void inorder(BSTNode r)
{ if (r !=
null)
{ inorder(r.getLeft());
System.out.print(r.getData() +" ");
inorder(r.getRight());
}
}
public void preorder()
{
preorder(root);
}
public void preorder(BSTNode r)
{ if (r !=
null)
{
System.out.print(r.getData() +" ");
preorder(r.getLeft()); preorder(r.getRight());
}
}
public void postorder()
{
postorder(root);
```

```
}
public void postorder(BSTNode r) {
if (r != null)
{
postorder(r.getLeft()); postorder(r.getRight());
System.out.print(r.getData() +" ");
}
}
}
import java.util.Scanner; public
class BinarySearchTree {
public static void main(String[] args)
{
@SuppressWarnings("resource")
Scanner scan = new Scanner(System.in);
BST bst = new BST();
System.out.println("Binary Search Tree Test\n"); char
ch;
do
{
System.out.println("\nBinary Search Tree Operations\n");
System.out.println("1. insert ");
System.out.println("2. delete");
System.out.println("3. search");
System.out.println("4. pre order");
System.out.println("5.In order");
System.out.println("6. post order");
int choice = scan.nextInt(); switch
(choice)
{ case 1
:
System.out.println("Enter integer element to insert");
bst.insert( scan.nextInt() );
break; case
2 :
System.out.println("Enter integer element to delete");
bst.delete( scan.nextInt() );
break; case
3 :
System.out.println("Enter integer element to search");
System.out.println("Search result : "+ bst.search( scan.nextInt() ));
break; case 4:
```

```
System.out.print("\nPre order : ");
bst.preorder(); break;
case 5:
System.out.print("\nIn order : ");
bst.inorder(); break; case  6:
System.out.print("\nPost order : ");
bst.postorder(); break; default :
System.out.println("Wrong Entry \n "); break;
}
System.out.println("\nDo you want to continue (Type y or n) \n");
ch = scan.next().charAt(0); } while (ch == 'Y'|| ch == 'y');
}
}
```

OUTPUTS

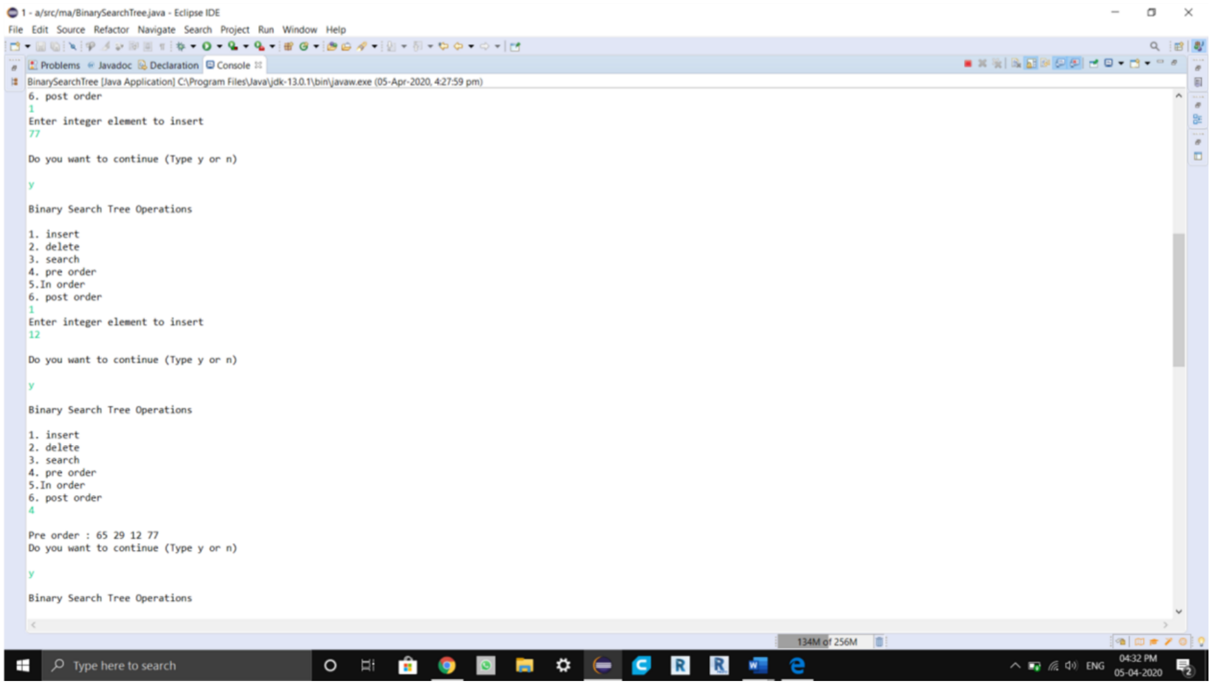
Screen Shots:



1 - a/src/ma/BinarySearchTree.java - Eclipse IDE

BinarySearchTree [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (05-Apr-2020, 4:27:59 pm)

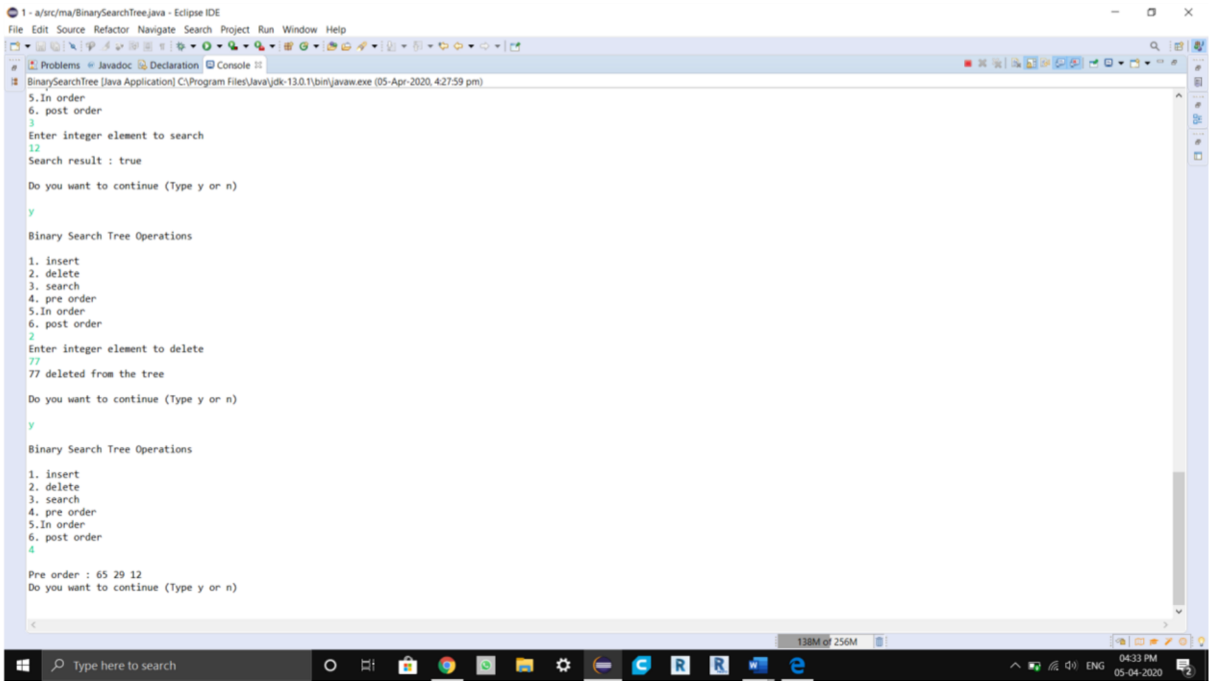
```
Binary Search Tree Test


Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
1
Enter integer element to insert
65

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
1
Enter integer element to insert
29

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
```



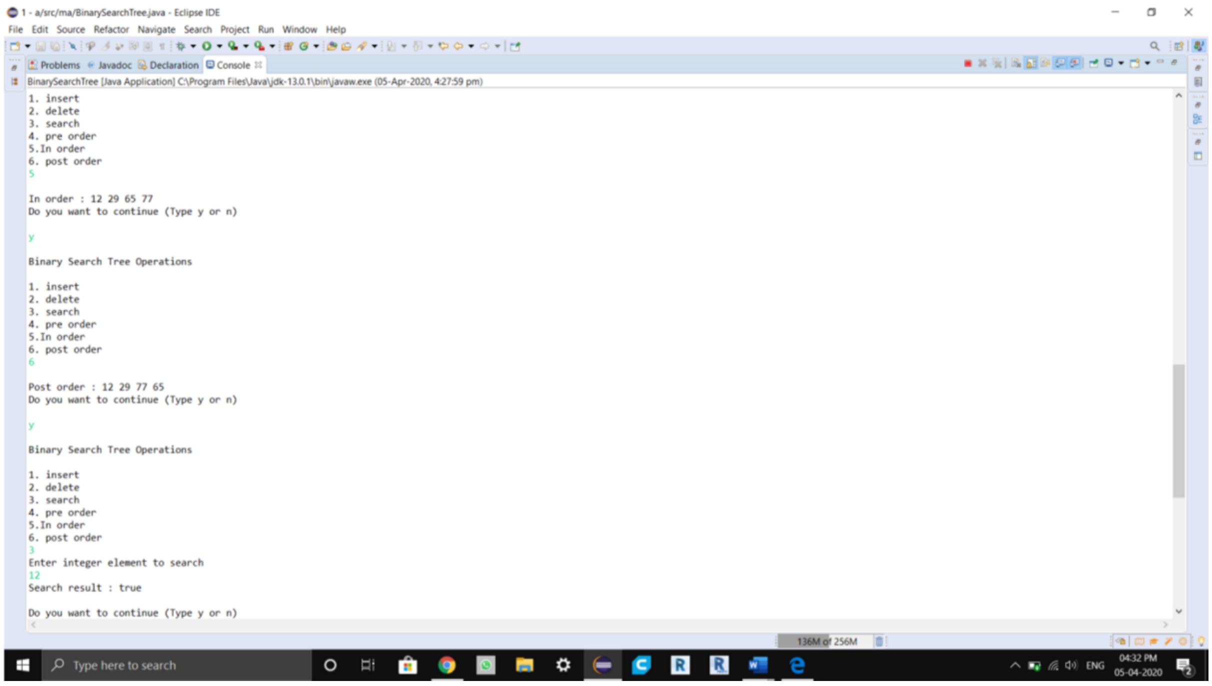
1 - a/src/ma/BinarySearchTree.java - Eclipse IDE

BinarySearchTree [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (05-Apr-2020, 4:27:59 pm)

```
6. post order
1
Enter integer element to insert
77

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
1
Enter integer element to insert
12

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
4

Pre order : 65 29 12 77
Do you want to continue (Type y or n)

y

Binary Search Tree Operations
```



```
5.In order
6. post order
3
Enter integer element to search
12
Search result : true

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
2
Enter integer element to delete
77
77 deleted from the tree

Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
4

Pre order : 65 29 12
Do you want to continue (Type y or n)
```



```
1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
5

In order : 12 29 65 77
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
6

Post order : 12 29 77 65
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. pre order
5.In order
6. post order
3
Enter integer element to search
12
Search result : true

Do you want to continue (Type y or n)
```

CONCLUSION

❾ By using this the program is very efficient and its code is easier than other data structures. There are left subtree and right subtree. The left subtree contains values that are less than the root node. However, the right subtree contains a value that is greater than the root node. BST is an advanced level algorithm that performs various operations based on the comparison of node values with the root node. Searches and inserts only touch a small amount of the items in the structure This means that there are less comparisons happening, and less work being done.