

**PROJECT REPORT**  
**Data Structure and Algorithms**  
**(UCS 406)**

*Word Finding using Binary Search Trees*

*by*

<b>ARHAM BANSAL</b>	<b>101603051</b>
<b>ARCHIT PURI</b>	<b>101603050</b>
<b>ARJUN GUPTA</b>	<b>101611008</b>



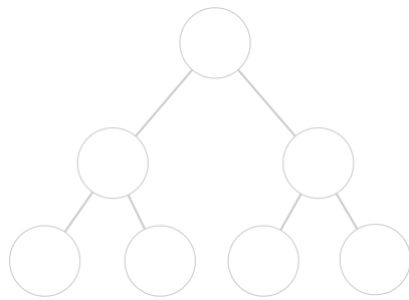
**THAPAR INSTITUTE**  
OF ENGINEERING & TECHNOLOGY  
(Deemed to be University)

Submitted to the  
**Computer Science & Engineering Department**  
**Thapar Institute of Engineering and Technology, Patiala**

**May 2018**

## Contents

Acknowledgement .....	3
1. Problem Formation .....	4
2. Analysis of problem.....	4
2.1 Analysis of Problem with Respect to Data Structures .....	4
2.2 Analysis of Problem with Respect to Algorithms .....	7
3. Flowchart .....	9
4. Results and Conclusions .....	10
4.1 Using Time Complexity.....	10
4.2 Using Space Complexity.....	11
4.3 Output.....	11
5. References.....	14



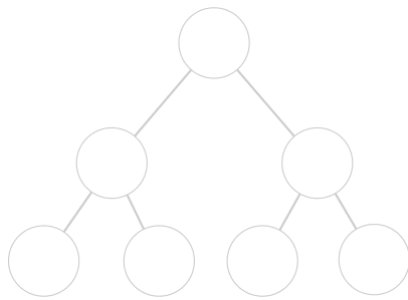
## Acknowledgement

This project would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to Dr. Karun Verma for his guidance and constant supervision as well as for providing necessary information regarding the project and for his support in completing the project.

We would like to express our gratitude towards our parents & classmates of for their kind co-operation and encouragement, which help us in completion of this project.

We would like to express our special gratitude and thanks to industry persons involved for giving us such attention and time and to our colleague in developing the project and to the people who have willingly helped us out with best of their abilities.



## 1. Problem Formation

Here in this project we have simply created a simple word-finding algorithm where we can search, modify and delete the words embedded into the Data Base. We have used the Data Structures to access the Database through the various Tree Functions. The most Elite feature of this project is that the user can also add the new words in the Database.

## 2. Analysis of problem

This is a Java Program to perform dictionary operations in binary search tree. In computer science, a binary search tree (BST), sometimes also called an ordered or sorted binary tree, is a node-based binary tree data structure where each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left sub-tree and smaller than the keys in all nodes in that node's right sub-tree. Each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree.

### 2.1 Analysis of Problem with Respect to Data Structures

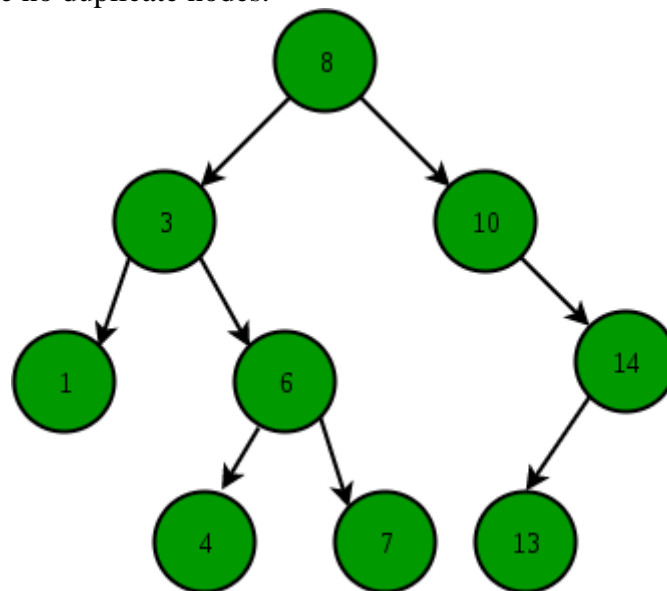
Binary Search Tree | Set 1 (Search and Insertion)

The following is definition of Binary Search Tree(BST) according to Wikipedia

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

There must be no duplicate nodes.



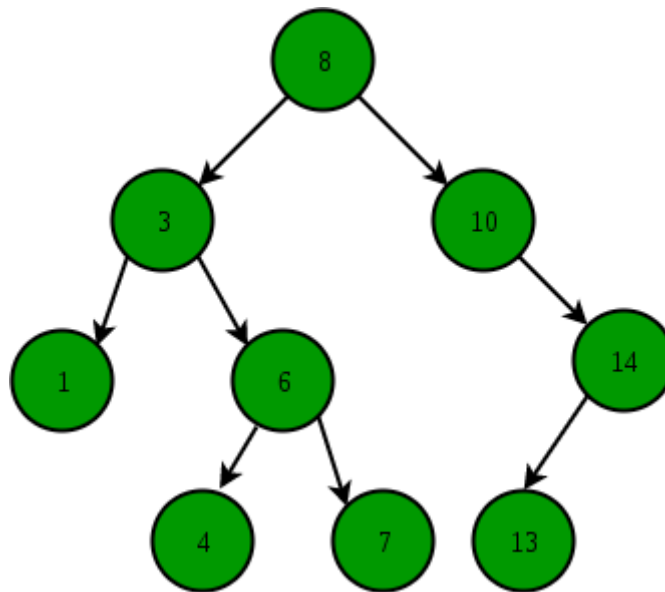
The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

### Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.

#### **Illustration to search 6 in below tree:**

1. Start from root.
2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.



### Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

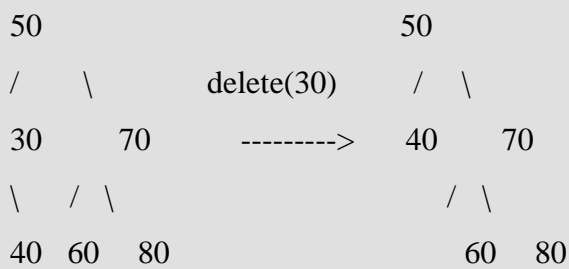
#### **Deletion:**

- 1) *Node to be deleted is leaf:* Simply remove from the tree.

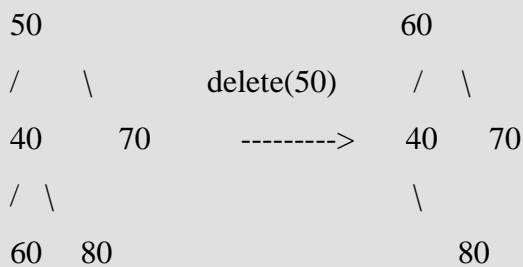
50				50			
/	\			/	\		
30		70		delete(20)	30	70	
/	\	/	\		/	\	
20	40	60	80		40	60	80

----->

2) **Node to be deleted has only one child:** Copy the child to the node and delete the child

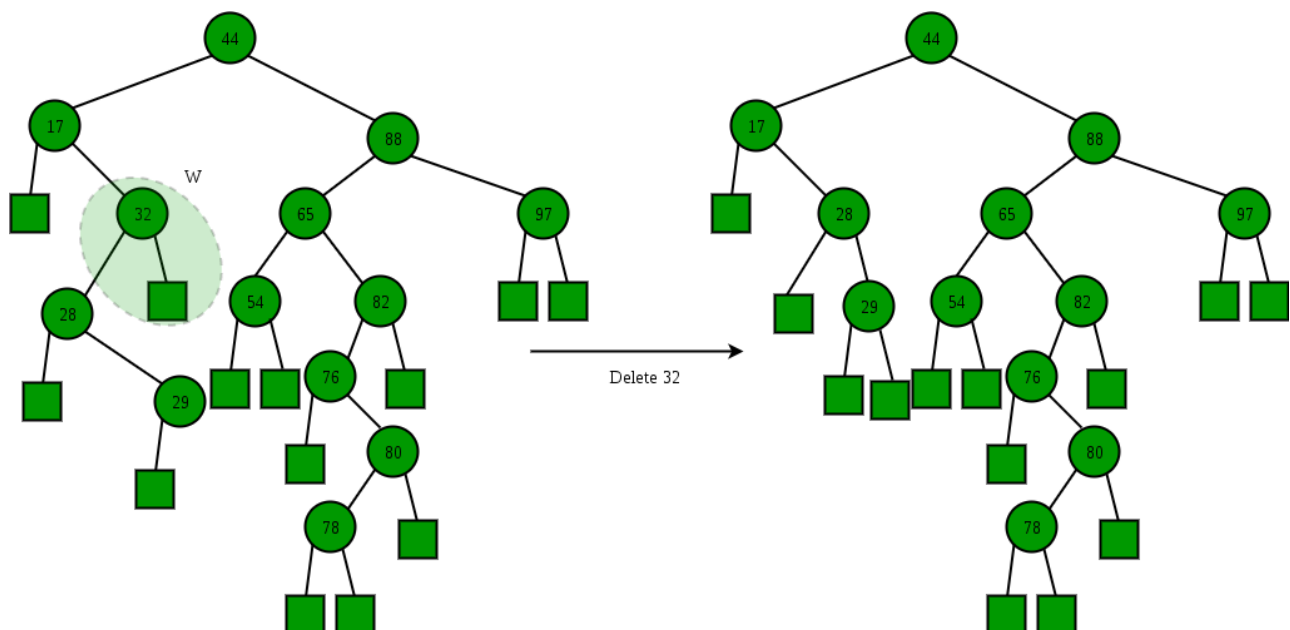


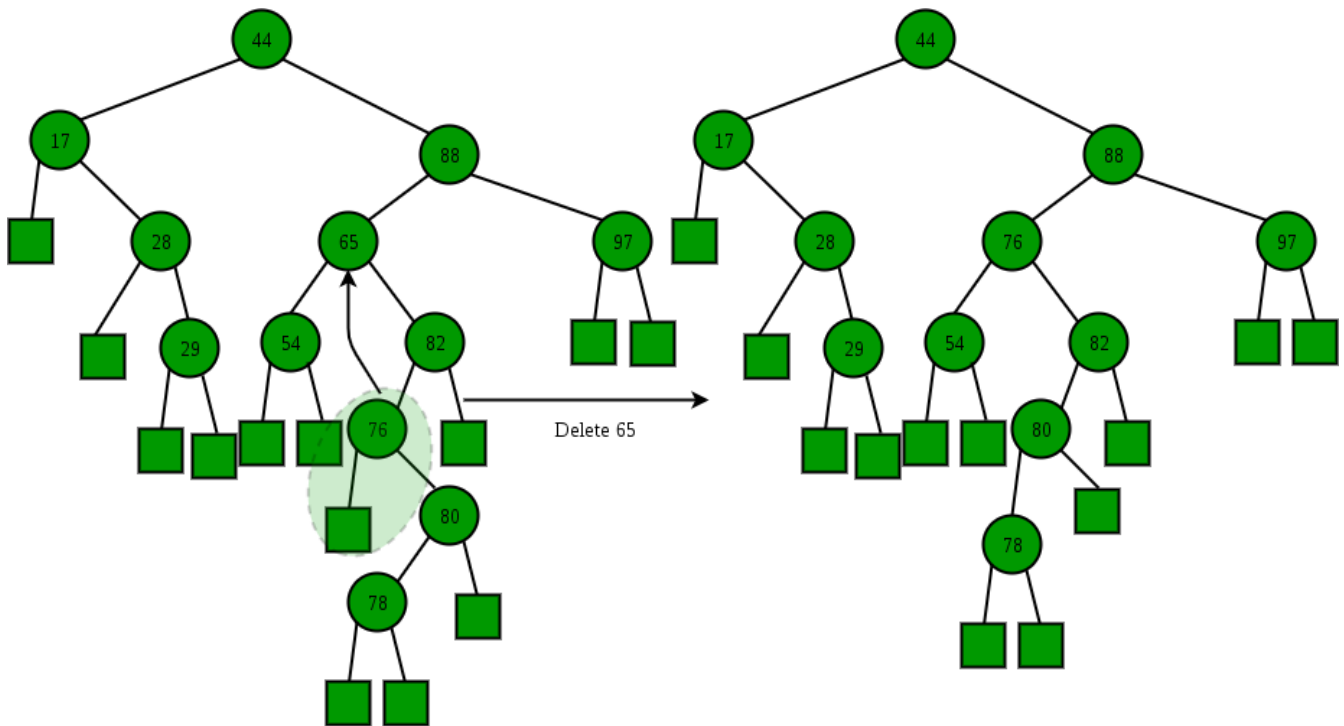
3) **Node to be deleted has two children:** Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.



The important thing to note is, inorder successor is needed only when right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in right child of the node.

### Illustration:





## 2.2 Analysis of Problem with Respect to Algorithms

### Searching a key

```
// A utility function to search a given key in BST
public Node search(Node root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == null || root.key == key)
        return root;
```

```
// val is greater than root's key
if (root.key > key)
    return search(root.left, key);
```

```
// val is less than root's key
return search(root.right, key);
}
```

### Insertion of a key

```
/* A recursive function to insert a new key in BST */
Node insertRec(Node root, int key) {
```

```
/* If the tree is empty, return a new node */
if (root == null) {
    root = new Node(key);
    return root;
}
```

```
/* Otherwise, recur down the tree */
```

```

if (key < root.key)
root.left = insertRec(root.left, key);
else if (key > root.key)
root.right = insertRec(root.right, key);

```

```

/* return the (unchanged) node pointer */
return root;
}

```

Deletion:

```

// This method mainly calls deleteRec()
void deleteKey(int key)
{
root = deleteRec(root, key);
}

```

```

/* A recursive function to insert a new key in BST */
Node deleteRec(Node root, int key)
{

```

```

/* Base Case: If the tree is empty */
if (root == null) return root;

```

```

/* Otherwise, recur down the tree */
if (key < root.key)
root.left = deleteRec(root.left, key);
else if (key > root.key)
root.right = deleteRec(root.right, key);

```

```

// if key is same as root's key, then This is the node
// to be deleted
else

```

```

{
// node with only one child or no child
if (root.left == null)
return root.right;
else if (root.right == null)
return root.left;

```

```

// node with two children: Get the inorder successor (smallest
// in the right subtree)
root.key = minValue(root.right);

```

```

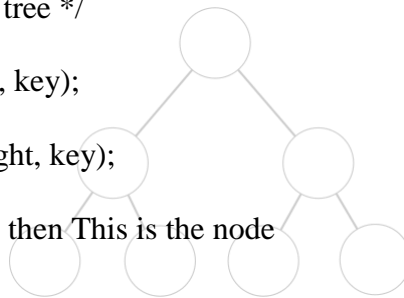
// Delete the inorder successor
root.right = deleteRec(root.right, root.key);
}
return root;

```

```

}

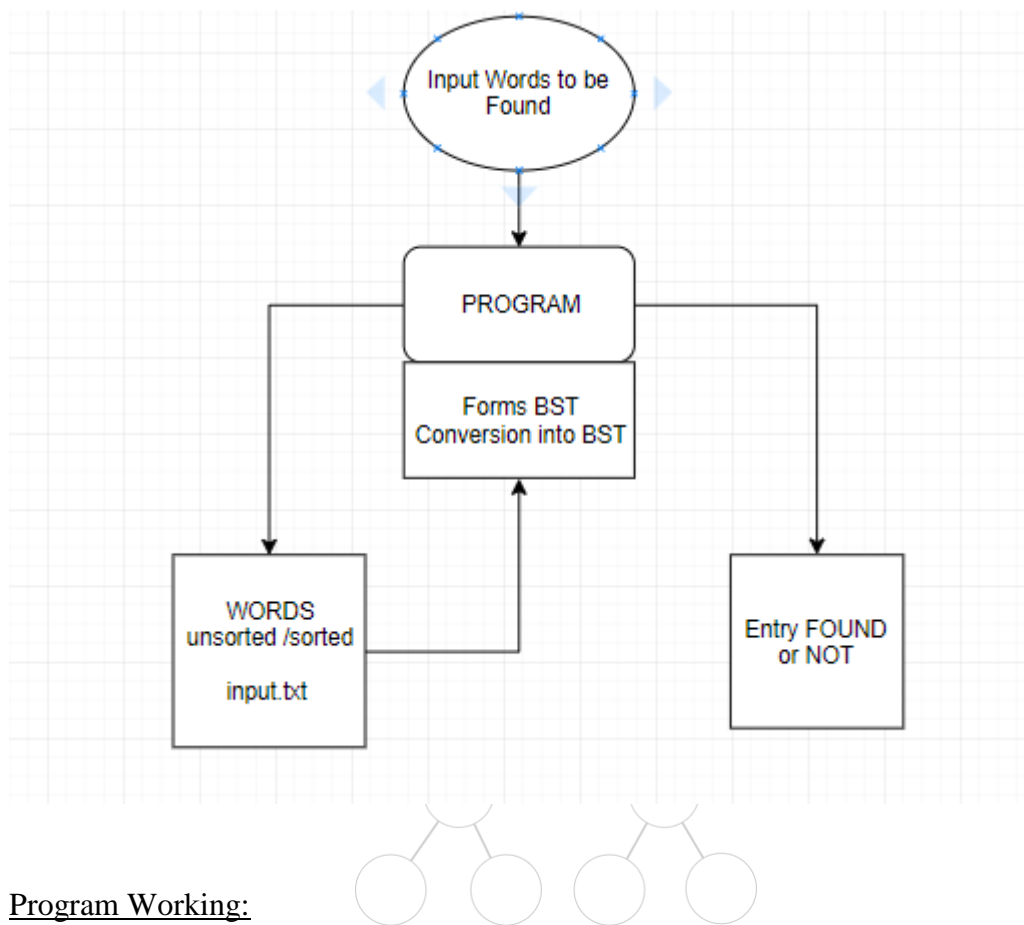
```



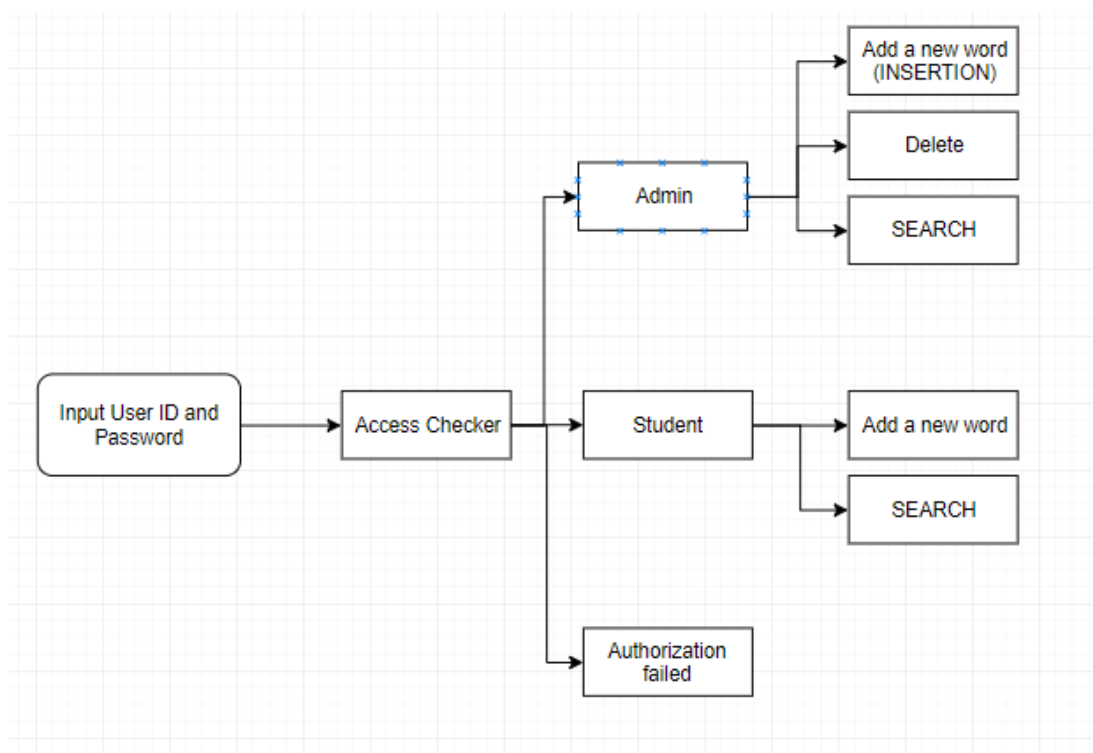


### 3. Flowchart

Main Program:



Program Working:



## 4. Results and Conclusions

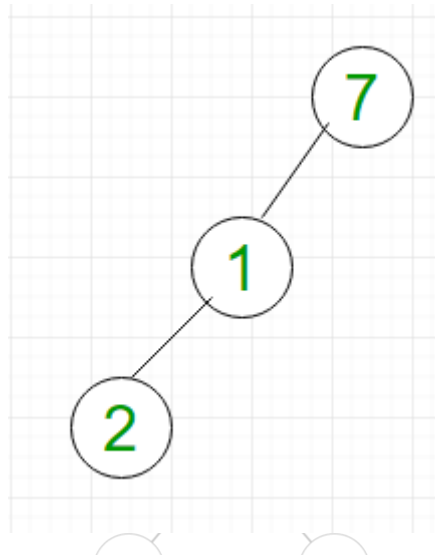
### 4.1 Using Time Complexity

#### Complexity of different operations in Binary tree, Binary Search Tree.

The main operations in binary tree are: search, insert and delete. We will see the worst case time complexity of these operations in binary trees.

#### Binary Tree –

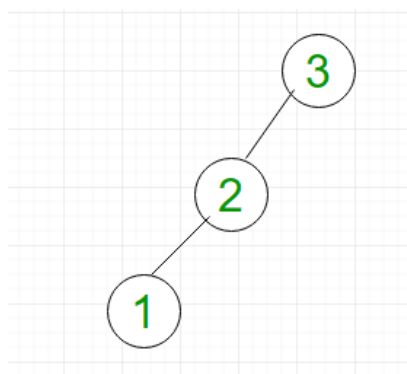
In a binary tree, a node can have maximum two children. Consider the left skewed binary tree shown in Figure 1.



- **Searching:** For searching element 2, we have to traverse all elements (assuming we do breadth first traversal). Therefore, searching in binary tree has worst case complexity of  $O(n)$ .
- **Insertion:** For inserting element as left child of 2, we have to traverse all elements. Therefore, insertion in binary tree has worst case complexity of  $O(n)$ .
- **Deletion:** For deletion of element 2, we have to traverse all elements to find 2 (assuming we do breadth first traversal). Therefore, deletion in binary tree has worst case complexity of  $O(n)$ .

#### Binary Search Tree (BST) –

BST is a special type of binary tree in which left child of a node has value less than the parent and right child has value greater than parent. Consider the left skewed BST shown in Figure 2.

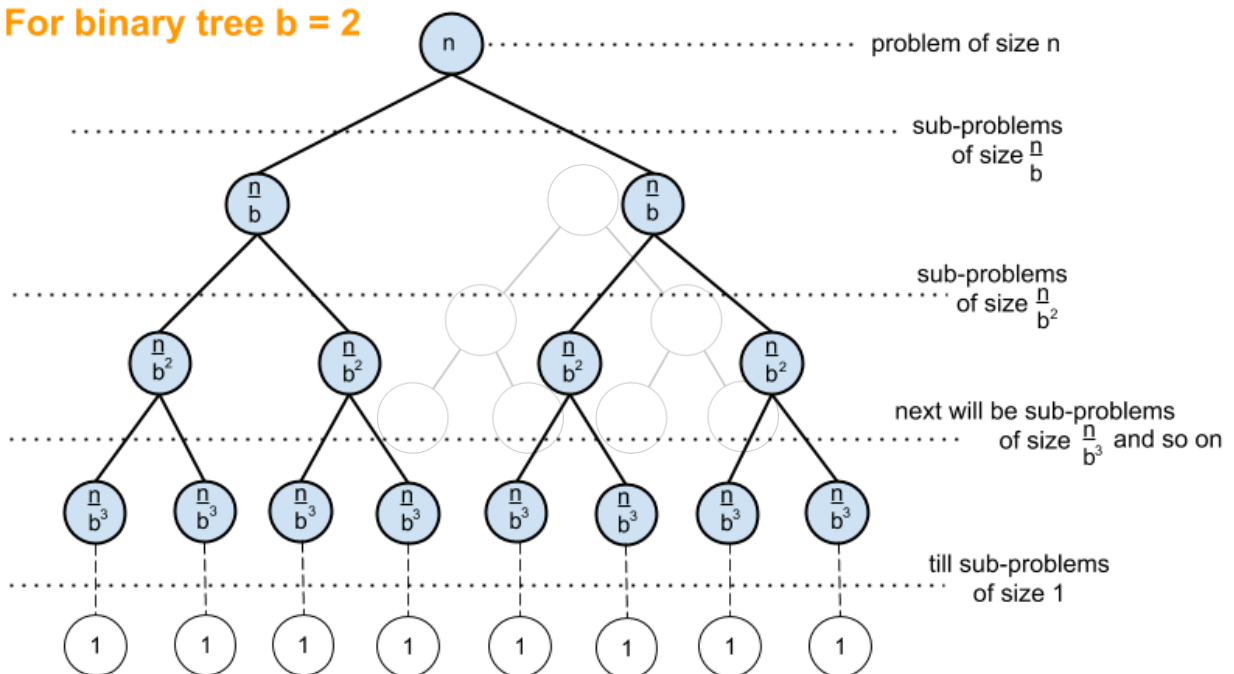


- **Searching:** For searching element 1, we have to traverse all elements (in order 3, 2, 1). Therefore, searching in binary search tree has worst case complexity of  $O(n)$ . In general, time complexity is  $O(h)$  where  $h$  is height of BST.
- **Insertion:** For inserting element 0, it must be inserted as left child of 1. Therefore, we need to traverse all elements (in order 3, 2, 1) to insert 0 which has worst case complexity of  $O(n)$ . In general, time complexity is  $O(h)$ .
- **Deletion:** For deletion of element 1, we have to traverse all elements to find 1 (in order 3, 2, 1). Therefore, deletion in binary tree has worst case complexity of  $O(n)$ . In general, time complexity is  $O(h)$ .

## 4.2 Using Space Complexity

The space complexity in BST for searching, insertion and deletion is  $O(n)$ .

**For binary tree  $b = 2$**



The height of the above tree is answer to the following question: How many times we divide problem of size  $n$  by  $b$  until we get down to problem of size 1?

The other way of asking same question:

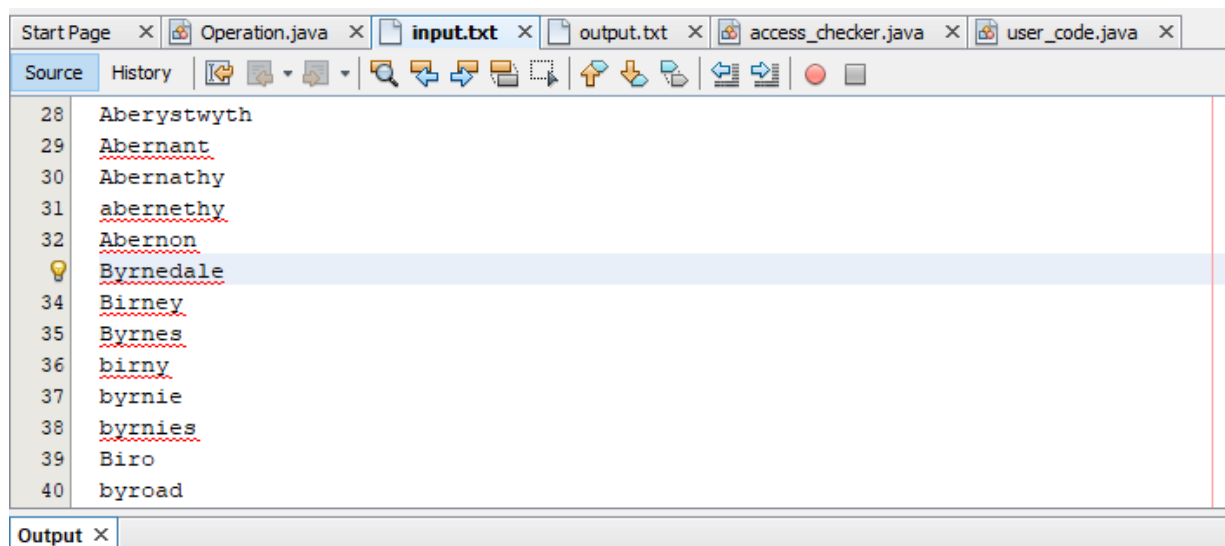
$$\text{when } \frac{n}{b^x} = 1 \quad [\text{in binary tree } b = 2]$$

$$\text{i.e. } n = b^x \text{ which is } \log_b n \quad [\text{by definition of logarithm}]$$

## 4.3 Output

### 1. INSERTION

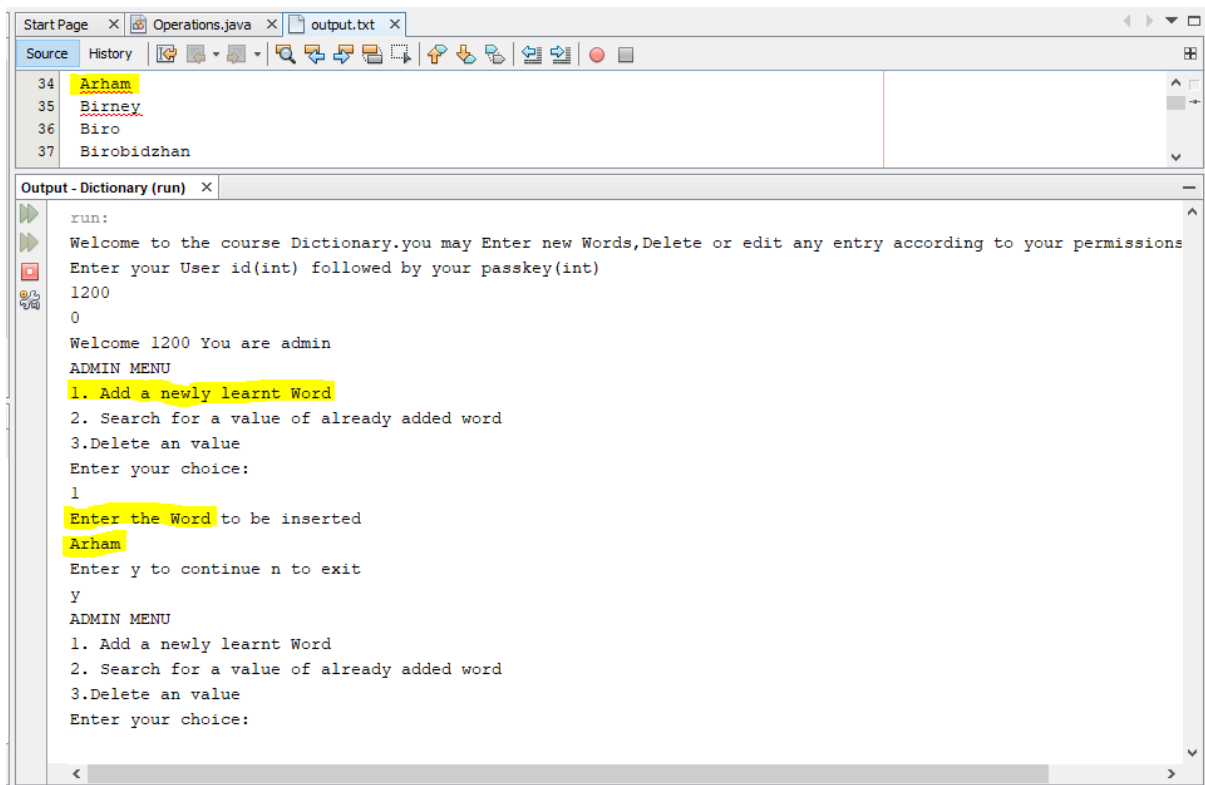
Input File Snippet:



The screenshot shows an IDE window with several tabs: 'Start Page', 'Operation.java', 'input.txt', 'output.txt', 'access\_checker.java', and 'user\_code.java'. The 'Source' tab is active, displaying a list of words from line 28 to 40. The words are: Aberystwyth, Abernant, Abernathy, abernethy, Abernon, Byrnedale (highlighted with a light blue background), Birney, Byrnes, birny, byrnie, byrnies, Biro, and byroad. The 'Output' tab is also visible at the bottom.

```
28 Aberystwyth
29 Abernant
30 Abernathy
31 abernethy
32 Abernon
33 Byrnedale
34 Birney
35 Byrnes
36 birny
37 byrnie
38 byrnies
39 Biro
40 byroad
```

After Insertion Snippet:

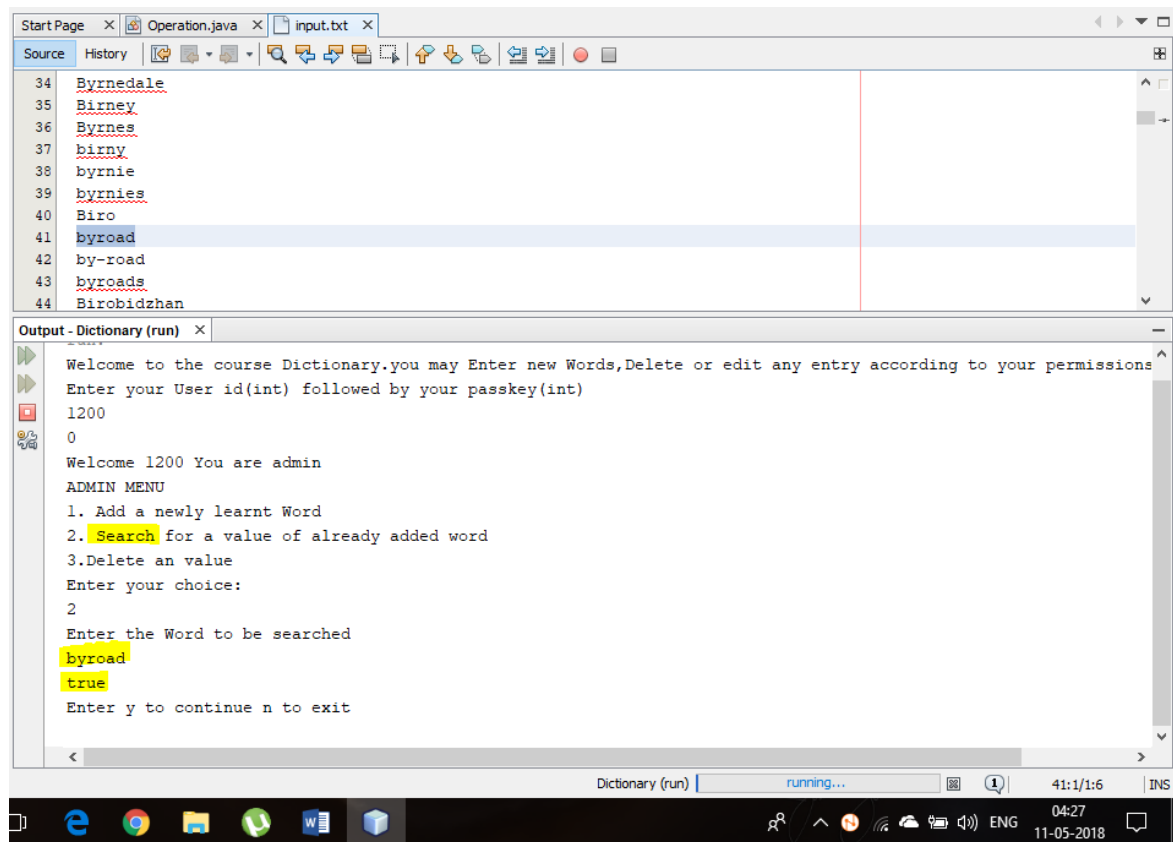


The screenshot shows an IDE window with tabs for 'Start Page', 'Operations.java', and 'output.txt'. The 'Source' tab is active, displaying a list of words from line 34 to 37: Arham (highlighted), Birney, Biro, and Birobidzhan. The 'Output - Dictionary (run)' tab is also visible, showing the output of the program. The output text is as follows:

```
run:
Welcome to the course Dictionary.you may Enter new Words,Delete or edit any entry according to your permissions
Enter your User id(int) followed by your passkey(int)
1200
0
Welcome 1200 You are admin
ADMIN MENU
1. Add a newly learnt Word
2. Search for a value of already added word
3.Delete an value
Enter your choice:
1
Enter the Word to be inserted
Arham
Enter y to continue n to exit
y
ADMIN MENU
1. Add a newly learnt Word
2. Search for a value of already added word
3.Delete an value
Enter your choice:
```

## 2. SEARCHING

### Case 1: True

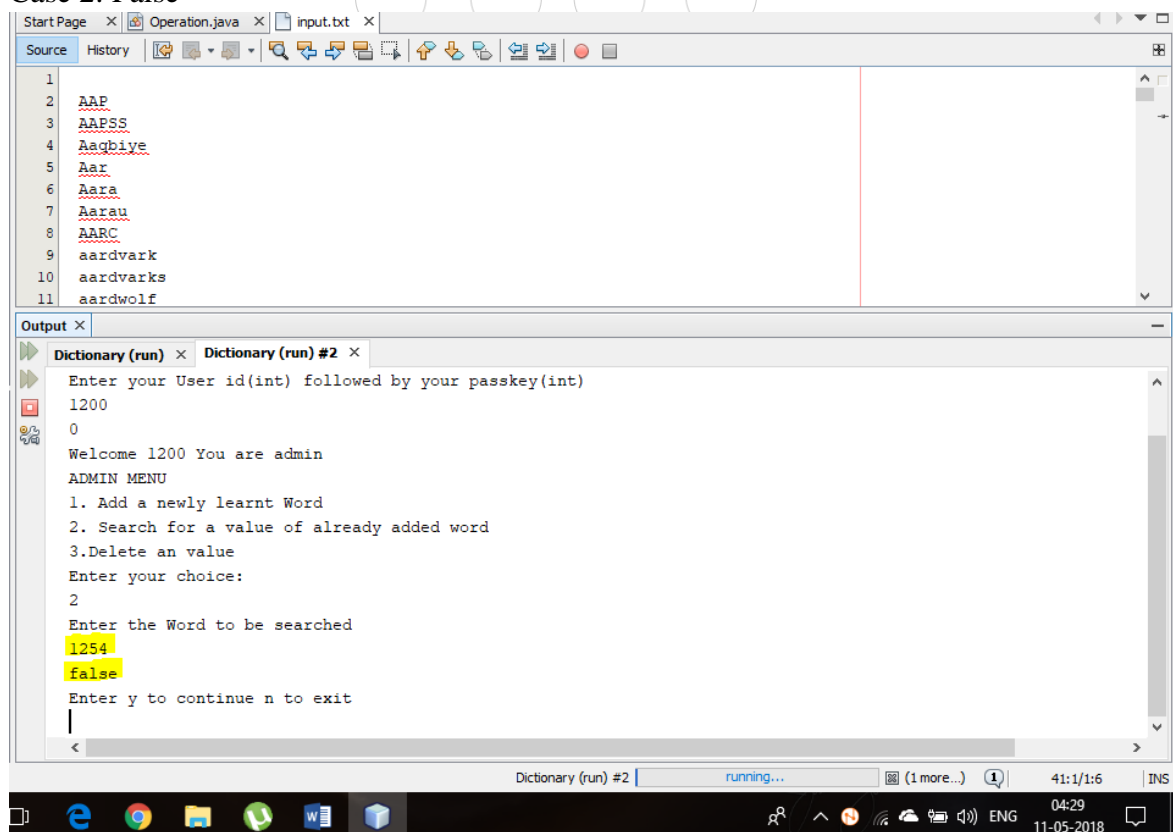


```
Start Page x Operation.java x input.txt x
Source History
34 Byrnedale
35 Birney
36 Byrnes
37 birny
38 byrnie
39 byrnies
40 Biro
41 byroad
42 by-road
43 byroads
44 Birobidzhan

Output - Dictionary (run) x
Welcome to the course Dictionary.you may Enter new Words,Delete or edit any entry according to your permissions
Enter your User id(int) followed by your passkey(int)
1200
0
Welcome 1200 You are admin
ADMIN MENU
1. Add a newly learnt Word
2. Search for a value of already added word
3.Delete an value
Enter your choice:
2
Enter the Word to be searched
byroad
true
Enter y to continue n to exit

Dictionary (run) running... 41:1/1:6 INS
```

### Case 2: False



```
Start Page x Operation.java x input.txt x
Source History
1
2 AAP
3 AAPSS
4 Aaqbiye
5 Aar
6 Aara
7 Aarau
8 AARC
9 aardvark
10 aardvarks
11 aardwolf

Output x
Dictionary (run) x Dictionary (run) #2 x
Enter your User id(int) followed by your passkey(int)
1200
0
Welcome 1200 You are admin
ADMIN MENU
1. Add a newly learnt Word
2. Search for a value of already added word
3.Delete an value
Enter your choice:
2
Enter the Word to be searched
1254
false
Enter y to continue n to exit

Dictionary (run) #2 running... (1 more...) 41:1/1:6 INS
```

### 3. Deletion

The top screenshot shows a Java IDE with the following source code:

```
1
2 AAP
3 AAPSS
4 Aaqbiye
5 Aar
6
```

The output window shows the following text:

```
run:
Welcome to the course Dictionary.you may Enter new Words,Delete or edit any entry according to your p
Enter your User id(int) followed by your paskey(int)
1200
0
Welcome 1200 You are admin
ADMIN MENU
1. Add a newly learnt Word
2. Search for a value of already added word
3.Delete an value
Enter your choice:
3
Enter the Word to be deleted
AAP
AAP deleted from the tree
Enter y to continue n to exit
|
```

The bottom screenshot shows the same source code with a blue oval highlighting the deletion logic in lines 11-12:

```
10
11
12
13 AAPSS
14 AARC
15 Aaqbiye
16 Aar
```

A callout box with the text "AAP DELETED" points to the highlighted area.

## 5. References

<https://www.geeksforgeeks.org>

[https://en.wikipedia.org/wiki/Data\\_structure](https://en.wikipedia.org/wiki/Data_structure)

[https://www.tutorialspoint.com/data\\_structures\\_algorithms](https://www.tutorialspoint.com/data_structures_algorithms)

<https://stackoverflow.com>