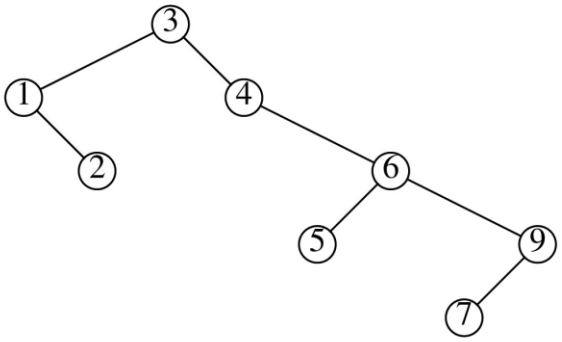
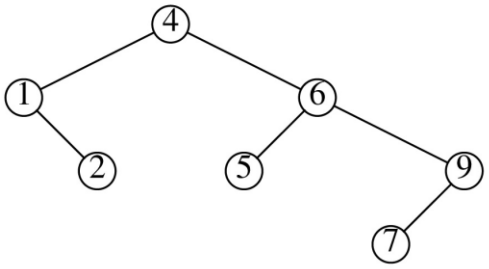
Assignment 4 - KEY

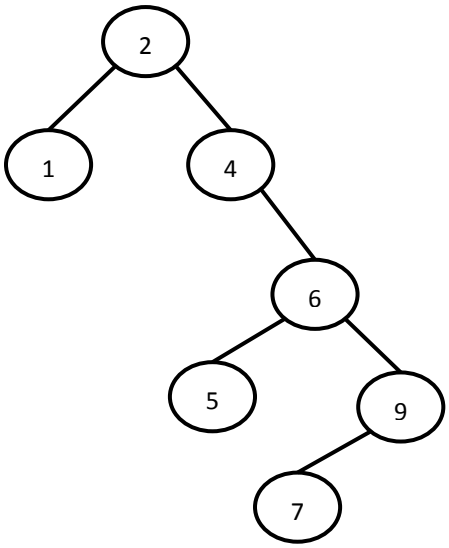
1.

a. Show the result of inserting 3,1,4,6,9,2,5, 7 into an initially empty binary search tree.
b. Show the result of deleting the root.





Alternate answer for 1b



2. Splay Tree

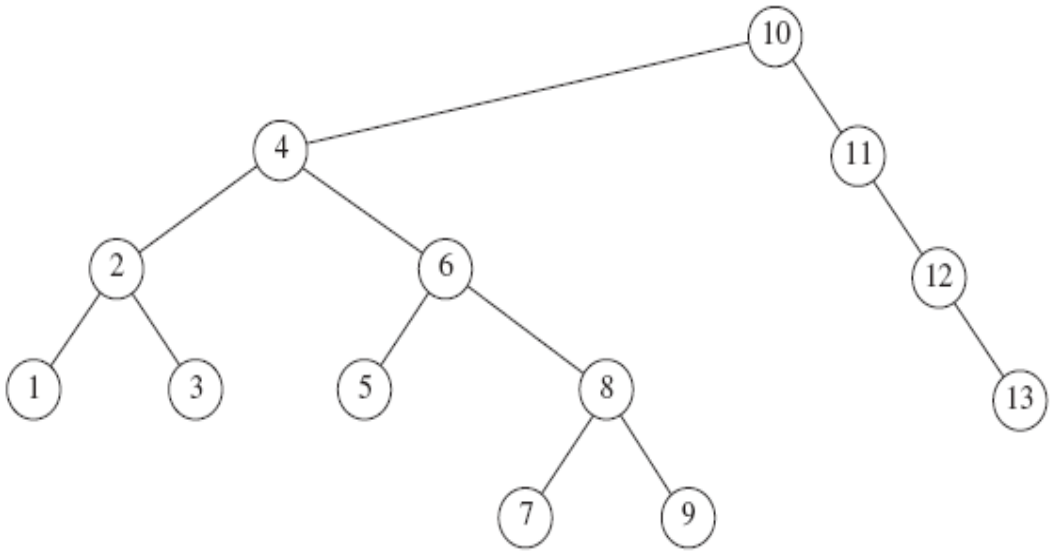
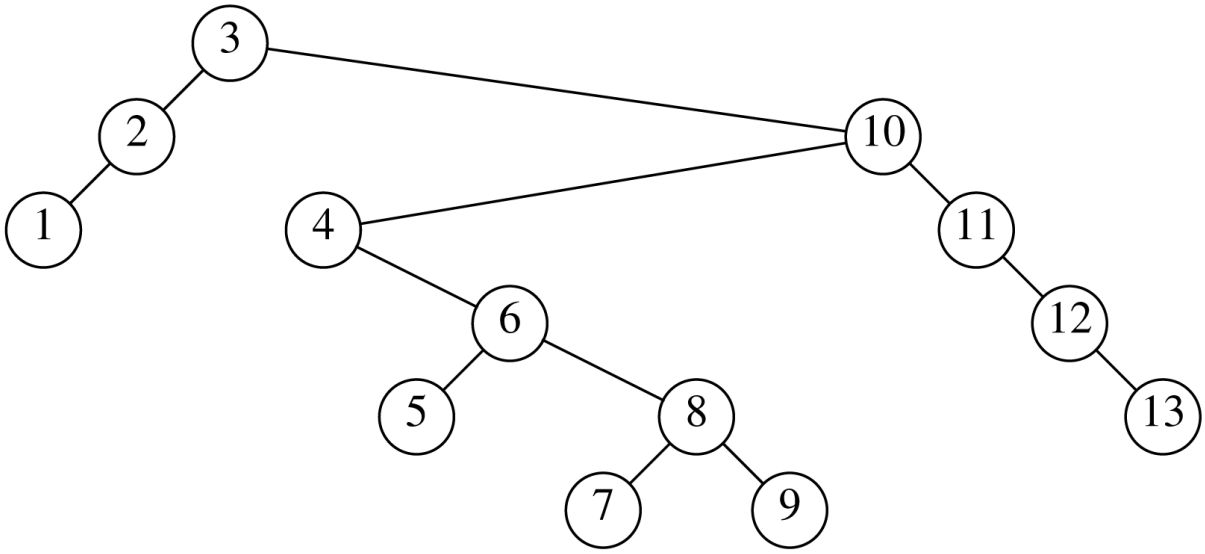


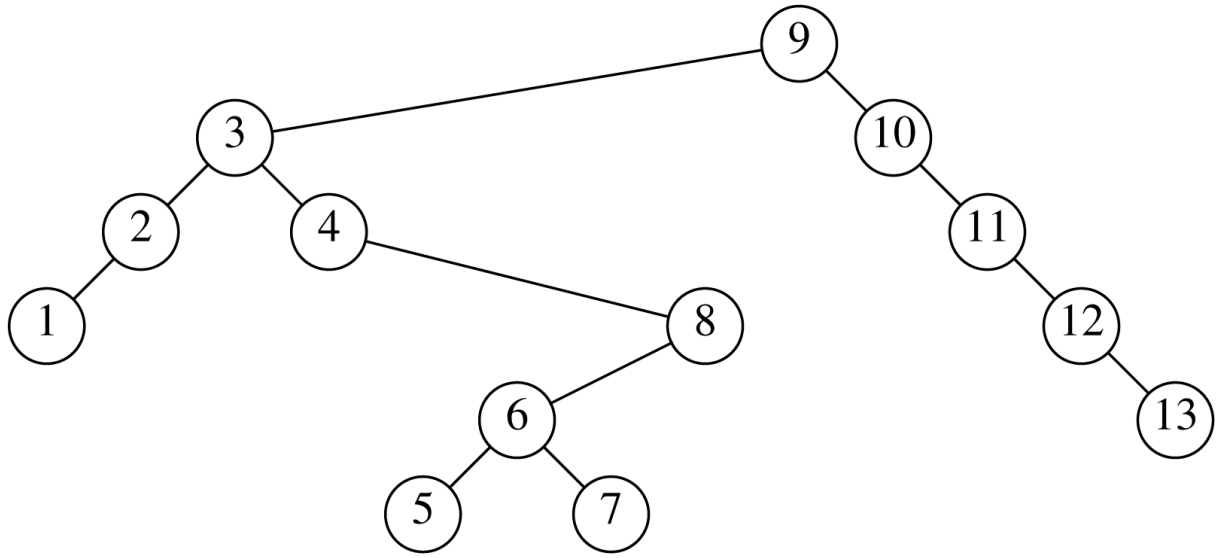
Figure 4.72 Tree for Exercise 4.27

a. Show the result of accessing the keys 3,9,1,5 in order in the splay tree

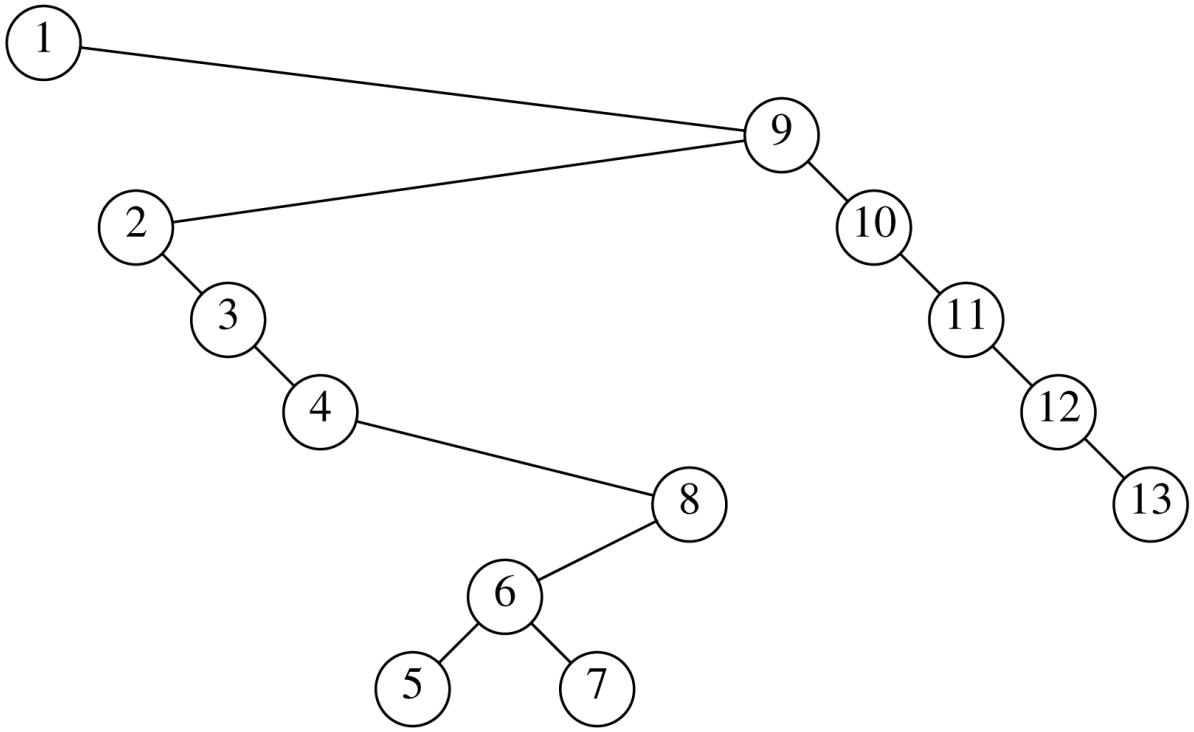
After accessing 3,



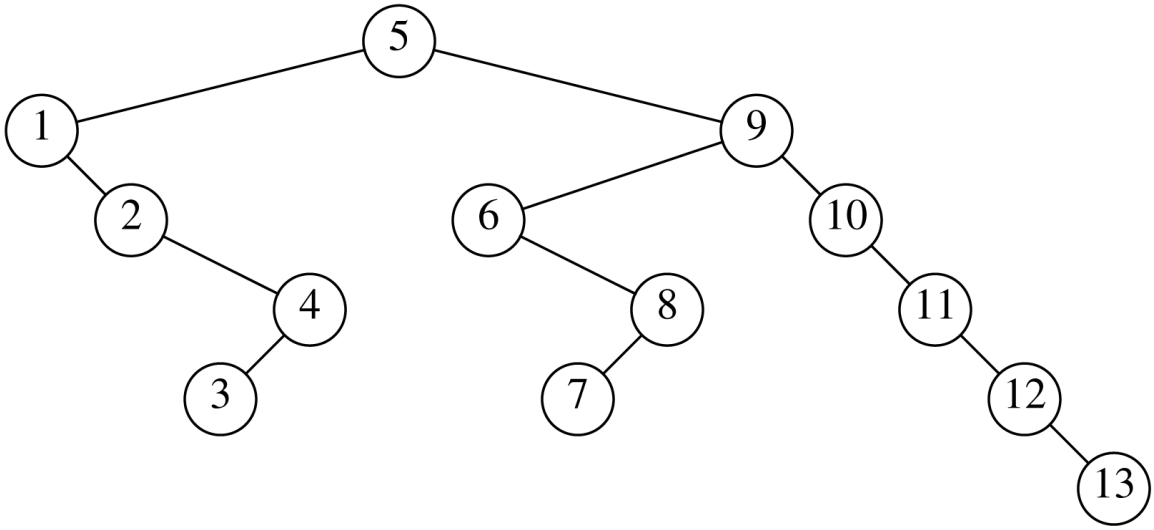
After accessing 9,



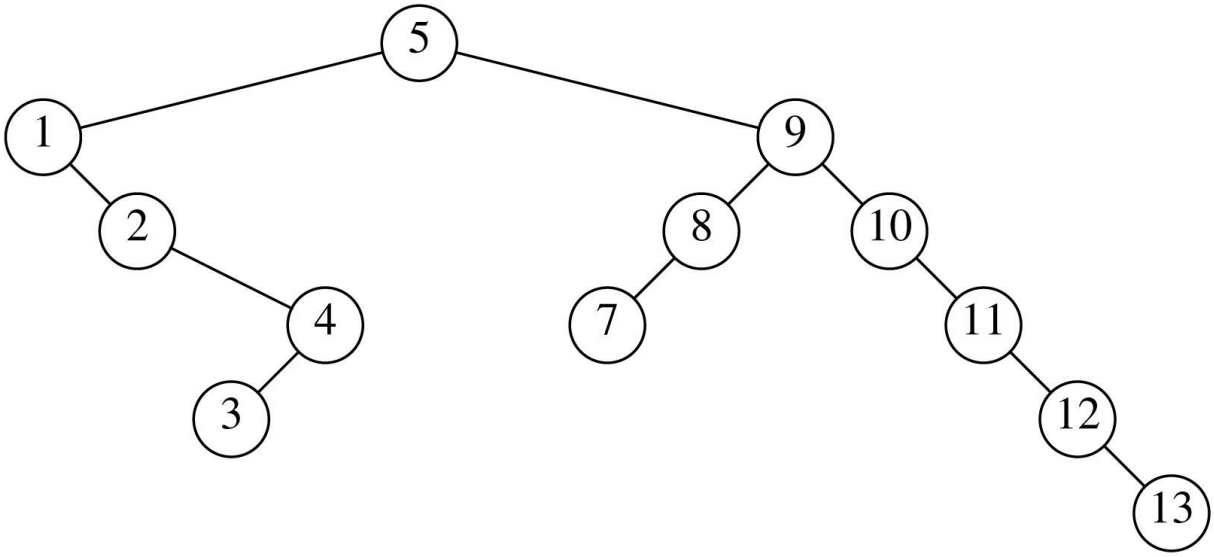
After accessing 1,



After accessing 5,



b. Show the result of deleting key 6.



3. Two binary trees are similar if they are both empty or both nonempty and have similar left and right subtree. Write a method to decide whether two binary tree are similar. What is the running time of your method?

The function shown here is clearly a linear time routine because in the worst case it does a traversal on both $t1$ and $t2$.

```
static boolean similar( Node t1, Node t2 )
{
    if( t1 == NULL || t2 == NULL )
        return t1 == NULL && t2 == NULL;
    return similar( t1.left, t2.left ) && similar( t1.right, t2.right );
}
```

4. **B-trees** [10 Points]

a. Given the following parameters:

1 Page on disk = 2048 bytes

Disk access time = 1milli-sec per byte

Pointer = 4 bytes

Key = 8 bytes

Data = 512 bytes per record (includes key)

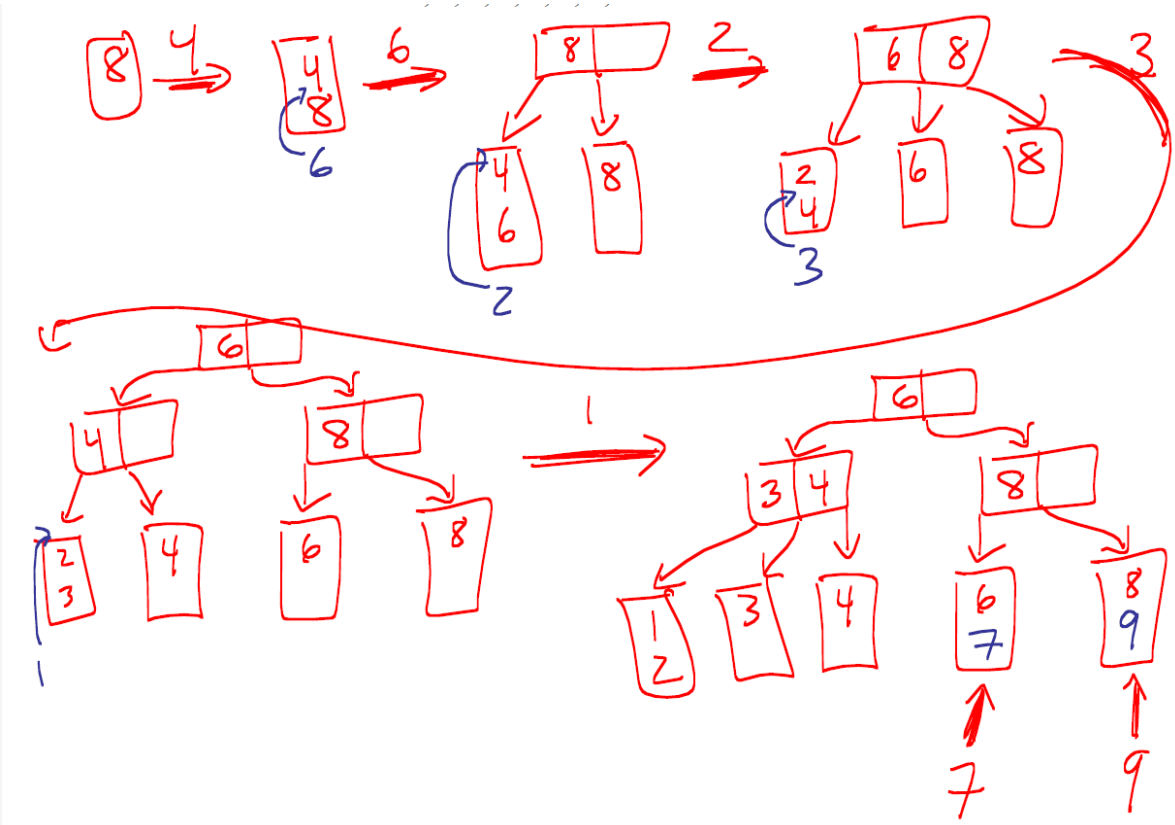
What are the best values for M and L.

M ➔ 4M + 8(M-1) = 2048 ➔ 12 M = 2056 ➔ M =171

L ➔ 2048/512 = 4

b. Insert the following values in this order, into a B-tree where M = 3 and L =2.

8, 4, 6, 2, 3, 1, 7, 9



5. Given a binary search tree and a value *k*, write a pseudo code to find a node in the binary search tree whose value is closest to *k*. [10 Points]

The routine takes as input a binary search tree (its root node) and a value the same generic type as the value K.

```
AnyType Closest(BST<AnyType> root, AnyType K) {
      if ( root == null) // Empty Tree
                        return null;


      root_difference = root.element.compareTo(K);
      if(root_difference == 0 ) return root.element;
```

//If the element at root is considered less to the value of K, either root is the closest element to K or the nearer element will be only to the right of the root.

```
      if(root_difference<0 )
      {
                  Closest_right =  Closest(root.right, K);
                  right_difference = Closest_right.compareTo(K);
                  if(absolute_value(right_difference) <
                                          absolute_value(root_difference))
                              return right.element;
                  else
                              return root.element;
   }
```

//If the element at root is considered more to the value of K, either root is the closest element to K or the nearer element will be only to the left of the root.

```
      if(root_difference > 0 )
      {
            Closest_left =  Closest(root.left, K);
            left_difference = Closest_left.compareTo(K);
```

```
            if(absolute_value(left_difference) <
                        absolute_value(root_difference))
                  return left.element;
            else
                  return root.element;
}
```