

1. Place the following numbers in a Binary Search Tree:

6, 11, 9, 7, 4, 5, 10, 2, 19, 28, 1

Draw the Binary Search Tree.

What is the height of the tree?

Use the **height** method in the **BinarySearchTree** class to verify your answer.

Write the numbers in the order displayed, based on the following traversals:
in-order, pre-order, post-order, and level-order

Which node could be placed in the root position, if the root node was deleted, so that all remaining nodes would not need adjustment?

Draw the tree with the above change.

Are there any other nodes that could replace the root node?

Using Tree terminology, how would you describe the node or nodes used?

2. Insert the following numbers into a binary tree, in the order given, maintaining a complete binary tree after each step.

6	11	9	7	4	5	10	2	19	28	1
---	----	---	---	---	---	----	---	----	----	---

Write the numbers in the order displayed, based on the following traversals:
in-order, pre-order, post-order, and level-order

Is this a heap?

If not, draw or write the steps necessary to place them in a heap.

Implement the **buildHeap** and **heapify** methods to verify your answers.

3. Use the following recursive insert method, and the code discussed last week, to implement the *BinarySearchTree<T>* class.

```
protected BinaryNode<T> insert(T d, BinaryNode<T> root){
    if(root == null)
        root = new BinaryNode<T>(d);
    else
        if(root.data.compareTo(d) > 0)
            root.left = insert(d, root.left);
        else
            root.right = insert(d, root.right);
    return root;
}
```

Create another class to test the implementation.

Create an instance of the *BinarySearchTree* class.

```
public class BinarySearchTree<T extends Comparable<T>> {
    }
}
```

Insert the numbers into the tree

Display the numbers by calling the appropriate in-order, pre-order, post-order and level-order traversal methods.

(**Note:** those methods require a reference to the root node, therefore they should be private. The root node should be private, and the class should not have an accessor method that returns a reference to the root node.)

4. Implement the *buildHeap* and *heapify* methods to verify your answers to number 2 above. Add the methods and the necessary array and method calls to the class in the number 3 above.

```

public static void buildHeap(int[] a,int size) {
    // start from last parent to first parent
    for(int i = size / 2 - 1; i >= 0; i--)
        heapify(a,i,size);
}
public static void heapify(int[] a, int i, int size) {
    int l = 2 * i + 1; // left child
    int r = 2 * i + 2; // right child
    int largest=i; // parent

    // find the larger of parent and left child
    if(l <= size-1 && a[l] > a[i])
        largest = l;
    else
        largest = i;

    // find the larger of parent and right child
    if(r <= size-1 && a[r] > a[largest])
        largest = r;

    // swap parent and larger child if necessary
    if(largest != i) {
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;
        // repeat heapify until all children are in a heap
        heapify(a, largest, size);
    }
}

```

Height method for BinarySearchTree:

```

private int height(BinaryNode<T> node) {
    if(node == null)
        return -1;
    return(Math.max(height(node.left), height(node.right)) + 1);
}

```