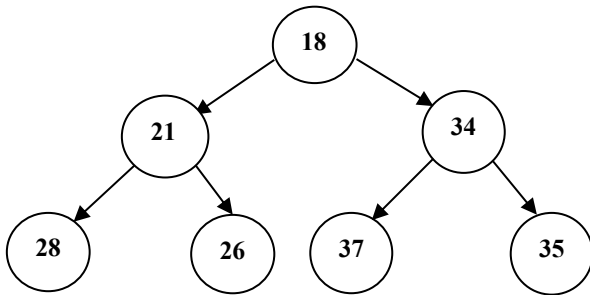**FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):**

STUDENT ID (e.g. 123456789):

INSTRUCTIONS:

Assume the following min-heap.

1. (4 pts) Draw the heap (as a tree) that would result from inserting **29** in the heap above.

2. (4 pts) Show how the heap above (the original, not the one from #1 with **29** in it) will look as an array (as discussed in class).

value							
index	0	1	2	3	4	5	6

Assume the code below with all necessary import statements.

```java
public class BinarySearchTree {
      private class Node {
            private int key;
            private Node left, right;

            private Node(int key) {
                  this.key = key;
            }
      }

      private Node root;
      private int minDist;
      private int nearestKey;

      public int getMinDist(){
            return minDist;
      }

      public int getNearestKey(){
            return nearestKey;
      }

      /*Assume code for add to add nodes to the BST as seen in class; smaller add to
      left, larger add to right, no duplicate keys allowed*/

      public void findNeighbor(int value) {   //assume tree with one node
            /* you will write this code and make one call to findNeighborAux*/
      }

}
```

_____

Sample Driver

```java
    public class SampleDriver {

        public static void main(String[] args) {

              String answer = "";

              BinarySearchTree tree = new BinarySearchTree();

              tree.add(40);tree.add(20);tree.add(60);
              tree.add(10);tree.add(30);tree.add(50);
              tree.add(70);tree.add(71);

              tree.findNeighbor(55);
              answer+=tree.getMinDist() +"\n";
              answer+=tree.getNearestKey() +"\n";
              tree.findNeighbor(56);
              answer+=tree.getMinDist() +"\n";
              answer+=tree.getNearestKey() +"\n";
              tree.add(52);
              tree.findNeighbor(56);
              answer+=tree.getMinDist() +"\n";
              answer+=tree.getNearestKey() +"\n";

              System.out.println(answer);

        }

    }
```

Driver Output
5
50
4
60
4
52

`findNeighbor` will perform any necessary non-recursive startup code and make one call to the private recursive `findNeighborAux`. When writing the code for `findNeighborAux` you can have any 2 parameters of your choice, but the code has to be recursive. The end result should be that <u>the field </u>`nearestKey` will be assigned the *nearest neighbor* to `value` (the argument passed into `findNeighbor`) **and** <u>the field</u> `minDist` will be the distance from `value` to the *nearest neighbor* (i.e. absolute difference).

The *nearest neighbor* to value is defined to be an actual key that is in the tree with smallest absolute difference between it and `value`. You can use `Math.abs` method to take the absolute value of an integer. When writing the code, you can assume the tree has at least one key in it, so there will always be a *nearest neighbor* to any value. Notice it is possible that 2 keys may tie to be the *nearest neighbor* to value. For example, in the driver code, both 50 and 60 have the smallest absolute difference of 5 to the value 55. In such a case, assign to `nearestKey` the smaller of the two (i.e. 50 instead of 60). When writing you code, you cannot have any loops and you cannot use any methods from the Java library other than the `Math.abs` method.