QUESTION 30

What is output by the code to the right?

A. 20a5

B. 4427

C. 3581

D. 268b

E. 3027

```
int x = Integer.parseInt("cff", 18);
int y = Integer.parseInt(
        "100102", 3);
String s = Integer.toString(
        x + y, 12);
out.println(s);
```

QUESTION 31

1

String s = "Hello,IamBob";

String[] r = s.split("\\p{Lu}");

out.println(r[2]);

This prints “am”

String str = "helloWorld";

String[] arr = str.split("\\p{Ll}");

System.***out***.println(Arrays.*toString*(arr)); //[, , , , , W]

Stack<Integer> s = **new** Stack<>();

s.push(100);

s.add(212);

s.push(7);

s.add(0, 3);

System.***out***.println(s);//[3, 100, 212, 7]

s.pop();

System.***out***.println(s.pop());//212

```
List<Integer> ls;
ls = new ArrayList<>(){{
add(1); add(4); add(5);
remove(1); add(7);
}};
out.println(ls);
```

```
out.println( (7 - 10) % 3 );
```

```
// Java Program to generate all unique
// permutations of a string
import java.util.*;

class GfG {

  // Recursive function to generate
  // all permutations of string s
  static void recurPermute(int index, StringBuilder s,
                           List<String> ans) {

    // Base Case
    if (index == s.length()) {
      ans.add(s.toString());
```

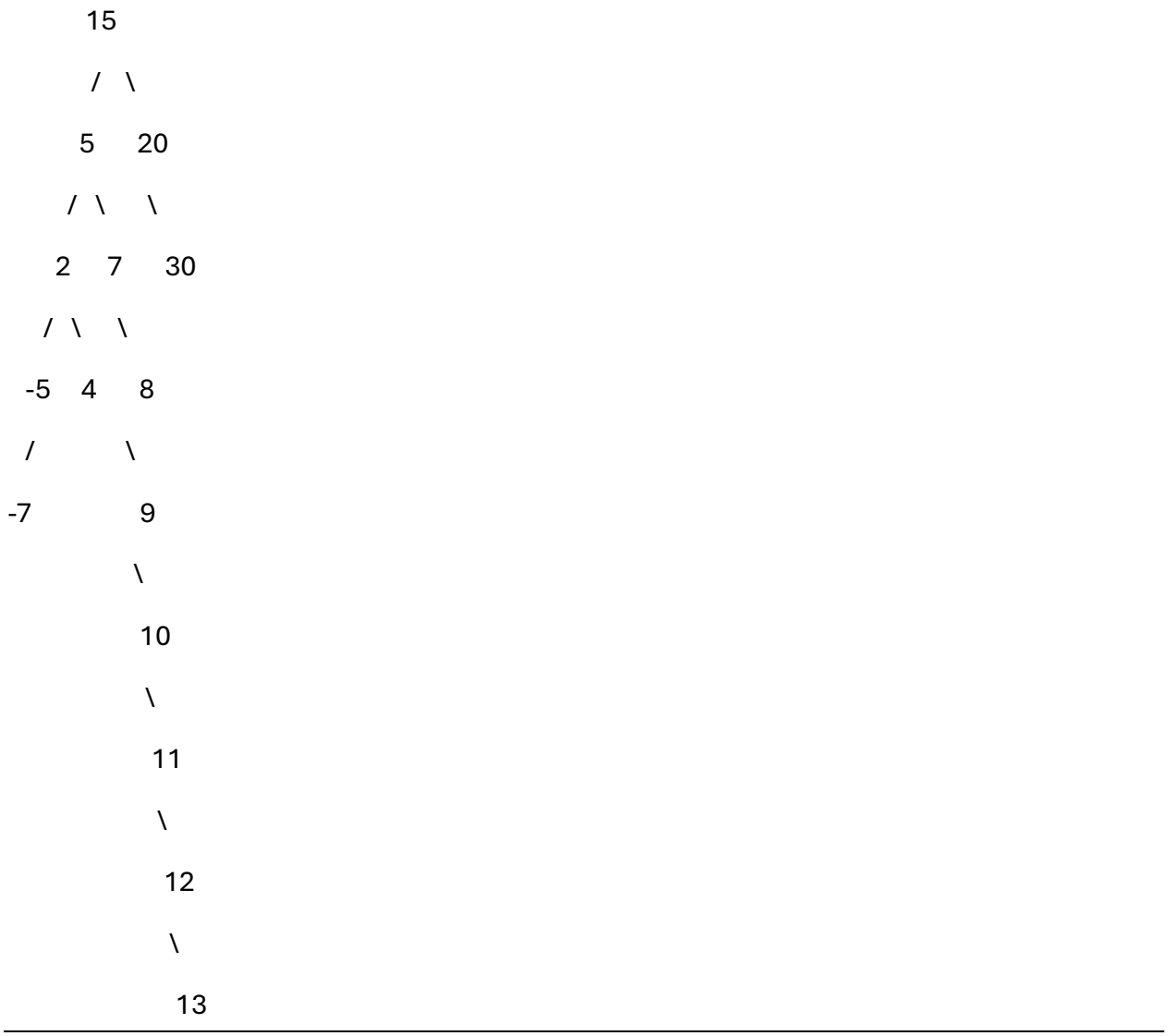
```
            return;
        }

        // Swap the current index with all
        // possible indices and recur
        for (int i = index; i < s.length(); i++) {
            swap(s, index, i);
            recurPermute(index + 1, s, ans);
            swap(s, index, i);
        }
    }

    // Swap characters at positions i and j
    static void swap(StringBuilder s, int i, int j) {
        char temp = s.charAt(i);
        s.setCharAt(i, s.charAt(j));
        s.setCharAt(j, temp);
    }

    // Function to find all unique permutations
    static List<String> findPermutation(String s) {

        // Stores the final answer
        List<String> ans = new ArrayList<>();
        StringBuilder str = new StringBuilder(s);
```

```
        recurPermute(0, str, ans);

        // sort the resultant list
        Collections.sort(ans);

        return ans;
    }

    public static void main(String[] args) {
        String s = "ABC";
        List<String> res = findPermutation(s);
        for (String x : res) {
            System.out.print(x + " ");
        }
    }
}
```

If the following items are inserted into a binary search tree in the following order, what is the diameter of the BST? The

diameter is the longest path between any 2 nodes in the tree. Count the edges not the nodes.

15 20 30 5 2 7 8 9 10 11 12 13 -5 -7 4



```
class D {
public int f(int x) {
return x + 1;
}
}
class E extends D {
public int f(int x, int y) {
return f(x) + y;
}
public int f(int x) {
return super.f(x) + x;
}
}
////////////////////////////////
//// Client Code
E e = new E();
out.println(e.f(5));
```

```
class A implements Comparable<A> {

  private String st;

  public A(String s) {
     st = s;
  }
  public int compareTo(A other) {
     int q = other.st.length();
     int r = st.length();

     return q == r ?
        st.compareTo(other.st) :
        q > r ? 1 : -1;
  }
  public String toString() {
     return st;
  }
}

////////////////////////////////
//// Client code
Queue<A> q = new PriorityQueue<>();
String[] s =
     new String[]{"Hello", "Bob",
     "Chicken", "Ralph", "A",
     "WaterBottle"};
for (String st: s) {
  q.add(new A(st));
}
while (!q.isEmpty()) {
  out.print(q.poll());
}
```

What is output by the code to the right?

A. ABobChickenHelloRalphWaterBottle

B. WaterBottleChickenHelloRalphBobA

C. WaterBottleRalphHelloChickenBobA

D. ABobRalphHelloChickenWaterBottle

E. HelloBobChickenRalphAWaterBottle

3. Typical custom class example

```
class Person implements Comparable<Person> {
   private String name;
   private int age;

   // Constructor, getters...

   @Override
   public int compareTo(Person other) {
      // Most common pattern: compare by one field first
      int nameCompare = this.name.compareTo(other.name);
      if (nameCompare != 0) {
         return nameCompare;         // different names → decide here
      }

      // Names are equal → tie-breaker with age
      return Integer.compare(this.age, other.age);
      // or: return this.age - other.age;  ← old style (can overflow!)
   }
}
```

```
class APLUS {
public static int x = 0;
public int f(int y) {
x += y;
if (y > 10) {
return 7;
}
return y + f(x + 1);
}
}
////////////////////////////////////
//// Client Code
APLUS f = new APLUS();
f.f(1);
out.println(f.x);
```

17 January Test

Start at back