

Read each question carefully. When writing code, note that not all questions require a complete class, or even a complete method.

When a method is called, who is responsible for ensuring that the postcondition is valid?

- A. The person who is using the program.
- B. The programmer who called the method.
- C. The programmer who wrote the method.
- D. The programmer who implemented the Java Runtime System.

Why does a run-time analysis usually count arithmetic and other operations instead of the number seconds required for the program to run?

The time required for the program to run partially depends on the hardware (computer) it runs on, which makes it difficult to compare one algorithm to another. It also doesn't provide any information regarding how well the program will scale. Counting operations as a function of the program's input provides a hardware independent way to directly compare different programs and also tells us how well (or poorly) it will scale

What are boundary values (with regards to testing programs)?

A boundary value is an input that is one step away (or on the edge of) causes the code to exhibit a different behavior. It is important to test a program at know boundary values as they are generally more likely to reveal programming errors.

Consider this code that creates some Location objects with coordinates x=10 and y=20:

```
Location a = new Location(10,20);
NamedLocation b = new NamedLocation(10,20,"blue team base");
...
```

After this code executes, what are the values of these boolean expressions?

a==b	False	a.equals(b)	True
b==a	False	b.equals(a)	False

Note: there is some sample code posted on the home page that demonstrates this.

Consider the Java code below & determine what the final output is:

```
int x=0;
for(int a=0; a<4; a++)
{
    for(int b=0; b<a; b++)
    {
        x++;
    }
}
System.out.println("x: " + x);
```

```
x=0
a=0, (b=0, inner loop finishes before it can even start)
a=1, b=0, x++           x=1
a=2, b=0,1, x++, x++   x=3
a=3, b=0,1,2, x++, x++,x++ x=6
a=4, outer loop finishes

final output:
x: 6
```

Write a method that takes an array of doubles & returns the smallest element. If the array is empty, return Double.NaN. Include documentation comments.

```
/**
 * finds smallest value in the array
 * @param array - the array to search
 * @return smallest value in array or Double.NaN if array is empty
 */
public double smallestValue(double[] array)
{
    if(array.length == 0)
        return Double.NaN;
    double smallestValue = array[0];
    for(int a=1; a<array.length; a++)
    {
        if(array[a] < smallestValue)
            smallestValue = array[a];
    }
    return smallestValue;
}
```

Consider the Bag implemented using an array to hold data. What happens if the **insert** method is activated, and the array is already full? What is required to allow this to work even if the array had a weird length of zero?

In general, we create a new array whose size is greater than the old array. We cannot just double the old array size as the old array might have had a size of zero, so we double it & add one. Note: we don't have to double it, but we must make sure to increase it by at least one.

Convert each time formula to the best possible big-O notation. Do not include any spurious constants in your big-O answer. Numerically order the entries from fastest to slowest.

Time required	Big O	Order
n	O(n)	1
5n	O(n)	2
1+2+3+...+n	O(n <sup>2</sup> )	3
n!	O(n!)	5
4n <sup>2</sup> + 6n <sup>3</sup>	O(n <sup>3</sup> )	4

In big O notation indicate the performance of the following operations of IntArrayBag:

```
// returns the current array capacity
// assume there is currently enough capacity
add(int item) O(1)

// returns the current array capacity
// assume there is currently NOT enough capacity
add(int item) O(n)

// count the number of times item is in the bag
countOccurrences(int item) O(n)

// get the total number of items in the bag
size() O(1)

// make a new bag identical to the combination of b1 & b2
union(IntArrayBag b1, IntArrayBag b2) O(capacity1 + capacity2)
```

Briefly describe the purpose of a **try/catch** block.

The try/catch block is intended to:

- 1) execute code that may result in an anticipated possible exceptions
- 2) respond to anticipated possible exceptions should they occur

For example, we may wish to call a method with certain parameters that we don't know how to check ourselves, but don't want the program to crash if they violate some precondition.

Give an example of a method that throws an Exception.

```
// questions didn't request documentation comments, so I'll skip 'em
public double smallestValue(double[] array)
{
    if(array.length == 0)
        throw new IllegalArgumentException("empty array!");
    double smallestValue = array[0];
    for(int a=1; a<array.length; a++)
    {
        if(array[a] < smallestValue)
            smallestValue = array[a];
    }
    return smallestValue;
}
```

What 2 things must a class have in order to be cloneable?

```
The class must implement Cloneable & it must have a .clone() method
```

Give an example of a clone method:

```
// questions didn't request documentation comments, so I'll skip 'em
public Object clone( )
{
    FruitSnack fs;

    try
    {
        fs = (FruitSnack) super.clone();
    }
    catch(CloneNotSupportedException e)
    {
        throw new RuntimeException("class doesn't implement Cloneable");
    }

    // note: if FruitSnack uses other objects as instance fields,
    // we'd need to clone them too...

    return answer;
}
```