CS 61B Discussion 1 Fall 2018

1 Javaian Rhapsody

Next to each line, write out what you think the code will do when it is run. Assume the `Singer` class exists and that the code below compiles.

Comments are above the line they describe.

```
1
2  /* Declare a variable of type String and assign it the value "no". In Java,
3     all variables must be declared before they are used. */
4  String disagree = "no";
5
6  /* Declare a variable of type int and assign it the value 7. */
7  int x = 7;
8
9  /* Declare a variable of type Singer and initialize it using the Singer
10    constructor with the argument "Queen" */
11 Singer queen = new Singer("Queen");
12
13 /* Checks if x is greater than 0; if so, subtract 1 from x, then call queen's
14    sing method with argument "no", then go back to the beginning of the loop.
15    queen object sings "no" 7 times. */
16 while (x > 0) {
17     x -= 1;
18     queen.sing(disagree);
19 }
20
21 /* Declares a variable of type String array and initializes it to hold three
22    Strings. */
23 String[] phrases = {"Oh", "mamma mia", "let me go"};
24
25 /* Prints "Oh" to standard output */
26 System.out.print(phrases[0]);
27
28 /* Declares variable i and initialize to 0 and checks if it is less than 3. If
29    so, print "mamma mia", then add one to i, then go back to the beginning of
30    the loop and re-check that i < 3. "mamma mia" is printed out 3 times. */
31 for (int i = 0; i < 3; i += 1) {
32     System.out.print(" " + phrases[1]);
33 }
34
35 /* Prints "let me go" to standard output. */
36 System.out.print(" " + phrases[2]);
```

2 Mystery

Below is a function (or method) called `mystery1`. It takes an array of integers called `inputArray` and an integer `k` as arguments and returns an integer.

```
1  public static int mystery1(int[] inputArray, int k) {
2      int x = inputArray[k];
```

```
    int answer = k;
    int index = k + 1;
    while (index < inputArray.length) {
        if (inputArray[index] < x) {
            x = inputArray[index];
            answer = index;
        }
        index = index + 1;
    }
    return answer;
}
```

Write the return value of `mystery1` if `inputArray` is the array {3, 0, 4, 6, 3} and `k` is 2. Then, describe in English what `mystery1` returns.

The `mystery1` function returns 4. `mystery1` returns the index of the smallest element that occurs at or after index `k` in the array. If `k` is greater than or equal to the length of the array or less than 0, an `ArrayIndexOutOfBoundsException` will be thrown at runtime.

The variable `x` keeps track of the smallest element found so far and the variable `answer` keeps track of the index of this element. The variable `index` keeps track of the current position in the array. The while loop steps through the elements of the array starting from index `k + 1` and if the current element is less than `x`, `x` and `answer` are updated.

Extra: Below is another function called `mystery2`. It takes an array of integers called `inputArray` as an argument and returns nothing.

```
public static void mystery2(int[] inputArray) {
    int index = 0;
    while (index < inputArray.length) {
        int targetIndex = mystery1(inputArray, index);
        int temp = inputArray[targetIndex];
        inputArray[targetIndex] = inputArray[index];
        inputArray[index] = temp;
        index = index + 1;
    }
}
```

Write what `mystery2` will do if `inputArray` is the array {3, 0, 4, 6, 3}. Then, describe in English what `mystery2` does.

`mystery2` doesn't return anything because its return type is `void`.

If `mystery2` is called on the array {3, 0, 4, 6, 3}, then after the method runs, the array will be {0, 3, 3, 4, 6}. Given any array, the method `mystery2` sorts the elements of the array in increasing order.

At the beginning of each iteration of the while loop, the first `index` elements of the array are in sorted order. Then the method `mystery1` is called to find the index of the smallest element of the array occurring at or after `index`. The element at the index returned by `mystery1` is then swapped with the element at position `index` so that the first `index + 1` elements of the array are in sorted order.

3 Fibonacci

Implement `fib1` recursively. `fib1` takes in an integer `N` and returns an integer representing the `N`th Fibonacci number. The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, . . . , where 0 is the 0th Fibonacci number.

```
public static int fib1(int N) {
    if (N <= 1) {
        return N;
    } else {
        return fib1(N - 1) + fib1(N - 2);
    }
}
```

Extra: Implement `fib2` in 5 lines or fewer that avoids redundant computation. `fib2` takes in an integer `N` and helper arguments `k`, `f0`, and `f1` and returns an integer representing the `N`th Fibonacci number. If you're stuck, try implementing `fib1` iteratively and then see how you can transform your iterative approach to implement `fib2`.

```
public static int fib2(int N, int k, int f0, int f1) {
    if (N == k) {
        return f0;
    } else {
        return fib2(N, k + 1, f1, f0 + f1);
    }
}
```

To compute the *N*th fibonacci number using `fib2`, call `fib2(N, 0, 0, 1)`.