Michael Lee
9/14/2020
112424954

1. Compute the tightest order, in big-O notation, of the following functions. In all cases, the variable to consider is ‘n’.

A. O(n^3) as n^3 ≥ n^2 for n ≥ 1 and n^3 ≥ - 0.1n for n ≥ 1. So if C = 5 and K = 1, Big O of f is O(n^3).

B. O(n) as n ≥ √n for n ≥ 1. So if C = 3 and K = 1, Big O of f is O(n).

C. O(1) as 1 ≥ sin(n) for n ≥ 1. So if C = 1 and K = 1, Big O of f is O(1).

D. O(√n) as i ≤ √n for n ≥ 1. So if C = 1 and K = 1, Big O of f is O(√n)

2. If f(n) = O(12n^2 + log(n) + 3), Big O of f is O(n^2). n^2 ≥ log(n) for n ≥ 3 and n^2 ≥ 3 for n ≥ 3. So, if C = 14 and K = 3, Big O of f is O(n^2).

If g(n) = O((2n^2 + 4n - 1)), Big O of g is also O(n^2). 4n^2 ≥ 4n for n ≥ 1 and n^2 ≥ -1 for n ≥ 1. So, if C = 7 and K = 1, Big O of g is O(n^2).

Therefore, Big O of f(n) and g(n) are equal.

3.

```
public boolean removeStudent(Student s) {
        try {                                                   L1
            if(studentIndex(s) >= 0) {                          L2
                int x = studentIndex(s);                        L3
                rawData[x] = null;                              L4
                for(int i = x; i < counter; i++) {              L5
                    rawData[i] = rawData[i+1];                  L6
                }
                rawData[counter--] = null;                      L7
                return true;                                    L8
```

```
                }
                return false;                                   L9
            } catch(NullPointerException e) {                   L10
                System.err.println(e.getMessage());             L11
                return false;                                   L12
            }
        }
```

L1 = 1. L2 = N → studentIndex function runs O(n) times worst case as it finds the index by searching through rawData. L3 = N. L4 = 1. L5 = N. L(6) = N-1. L7 = 1. L8 = 1. L9 = 1. L10 = 1. L11 = 1. L12 = 1.

So, if NOT NullPointerException, 1 + N + N + 1 + N + (N-1) + 1 + 1 → where studentIndex(s) >= 0.

1 + N + 1 → where studentIndex(s) < 0.

If NullPointerException, 1 + 1 + 1.

Therefore, the Big O of removeStudent for the worst case is O(n).

4.

```
public static double getMedian(int[] arr) {
        if(arr.length % 2 == 0) {                   // L1
            int a = arr.length / 2;                 //L2
            int b = (arr.length /2) - 1;            //L3
            return (arr[a] + arr[b]) / 2.0; //L4
        } else {
            return arr[arr.length / 2];     //L5
        }
 }
```

If arr.length % 2 = 0, then L1 = 1, L2 = 1, L3 = 1, L4 = 1. So, 1 + 1 + 1 + 1 = 4.

If arr.length % 2 != 0, then L1 = 1, L5 = 1. So 1 + 1 = 2.

Worst case for getMedian runs with 4 constants and thus has O(1).