EECS2030 FALL 2019

Lab 8

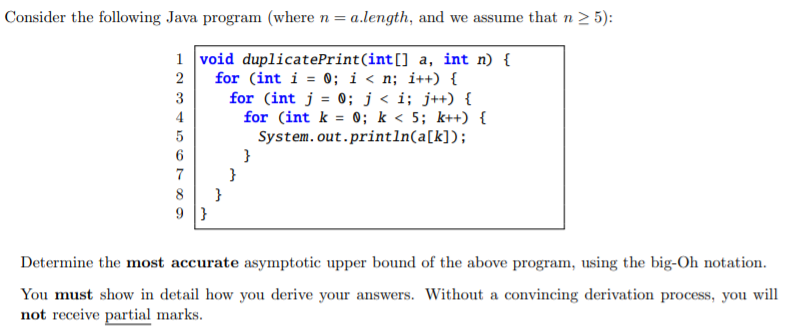
Analyzing the Time Complexity of a Program

Section B | Jackie Wang

**Krishaanth Manoharan**

**216463150**

Due Date: 11:59 PM, Wednesday, December 4, 2019

  
NOTE TO TAS **\*Primitive Operations can be ignored because they are of complexity O(1) such as int I = 0; a[i] = 0; etc. So anything I’ve listed as primitive operations are of O(1)\***

Line 1: The declaration of the void method, duplicatePrint. The parameters is a primitive integer array (a) and an integer (n) being a.length, with an assumption that n >= 5.

# Line 2:

We see that this line is a **for loop**.

Has **1 assignment (Primitive Operation)**, being int i = 0; (+1 times). Since this line is a for loop it must have a condition, and that being from i < n; which is **n comparisons**. Integer i increments by 1 every time it loops so (+n times). Total for this line:

i++ , will have **1 addition (Primitive Operation) + 1 assignment (Primitive Operation)** i = i + 1;

Need to also multiply by n – 1 because i < n:

# Line 3:

We see that this **for loop is nested under the line 2’s for loop**. So once line 2’s for loop stops looping, line 3’s loop will be prevented from further continuing

Has **1 assignment (Primitive Operation)**, being int j = 0; The condition of this for loop is j < i this will be more complex to calculate, so we can use a comparison chart.

**Comparison chart** for i (i < n) and j (j < i) execution values:

|  |  |  |
| --- | --- | --- |
| i = … | j < i | j = … |
| 0 | 0 < 0 | - |
| 1 | 0 < 1 | 0 |
| 2 | 0 < 2 | 0, 1 |
| 3 | 0 < 3 | 0, 1, 2 |
| 4 | 0 < 4 | 0, 1, 2, 3 |
|  |  |  |
| n - 1 | **0 < n - 1** | **0, 1, 2, 3, …, n -2** |
| n | **0 < n** | **0, 1, 2, 3, …, n-1** |

We found out that line 2’s for loop will reach +n times, thus integer i goes to n because at the end i++ will cause it to equal n. Looking at this chart we see that j goes to n-1 but at the last j++ it will cause the for loop to check again if j<i so it will reach **(+n times)**. Total for this line (including the primitive assignment int j = 0;):

j++ , will have **1 addition (Primitive Operation) + 1 assignment (Primitive Operation)** j = j + 1;

Need to also be multiplied by the number of iterations the loop is run.

# Line 4:

We see that this for loop is also nested by line 3’s for loop. The only significant difference here is that this for loops’ condition is k < 5. And k is initialized as 0 and then being k++ after each time this gets looped. Technically if we don’t do a precise calculation, we could say that this for loop acts like a complexity of O(1) (acts like primitive operations) because it loops ONLY 5 times, it doesn’t go to n times or gets affected by another variable so should be ignored.

Since we are doing a precise calculation, this for loop has **1 assignment (Primitive Operation)**, being int k = 0;(+1 times) The condition of this for loop is k < 5; and k++ so **k gets repeated 5 times so 5 comparisons**. k = 0, 1, 2, 3, 4. (+5 times)

Because this for loop is nested by line 3’s and then line 2’s for loop we multiply by the # of iterations the loop body is executed.

k++ , will have **1 addition (Primitive Operation) + 1 assignment (Primitive Operation)** k = k + 1;

Need to also multiply by 5 because of k < 5, and the number of iterations for line 3:

# Line 5:

This is just a print statement, where it prints the value contained in array a at index k. So with the assumption n >= 5 makes sense because if n is less than 5 that means a.length is less than 5. Which will cause an ArraysOutOfBoundsException once we call a[4] (or anything above a.length -1 as for index).

So, since this is just a print statement it is (+1 times) then indexing to an array another (+1 times).

Because this is inside the for loop by line 4 and nested by line 3 and line 2, we **multiply by the # of times the loop body is executed ( times because of line 4, times because of line 3, and times because of line 2.)**

**OVERALL POLYNOMIAL ADDED TOGETHER:**

This is **O() (Quadratic) because is the highest power in the polynomial .**

**Asymptotic Upper Bound Proof:**

Choosing as the coefficients sum,

Choosing n0 as such that,

Therefore, the given java program is bounded by **Time Complexity** **O().**