Sandeep Heera

Time Complexity Analysis

Snippet to analyze:

k = 0

for(i = 1; i <= n; i++) {

k = k + 3;

if(i > 10) {

for(j = 1; j <= i; j++) {

k = k +5;

}

}

Due to the inner loop which checks to see if *i* > 10, there will be two separate functions that determine the value of *k* following the execution of the code. The closed result will be a piecewise function and the determining factor will be whether or not *n* <= 10.

For *n* <= 10:

Since *k* starts at 0 and we’re taking the value of a previous *k* and adding 3 to it, the summation ends up being:

Which reduces to *k* = 3*n*.

For *n* > 10:

*k* will be equal to the above solution plus another summation which involves the execution of the inner loop. The indices of the summation will be from *i* = 11 to *n* since the loop only executes if

*i* > 10 and as before since we’re just adding a constant from *j* = 1 to *i* the summation becomes:

The summation after the 3*n* term can also be rewritten as:

So, the final value of *k* for *i* > 10 will be:

**The final closed piecewise function is:**

Time complexity analysis:

Assuming each operation takes constant time, the asymptotic running time of the code fragment for *n* <= 10 will just be *n* since the *k* = *k* + 3 is the only operation being executed. The summation would be:

The code fragment would thus have O(*n*) time complexity with positive constants *c0* and *n0* equal to 1 since:

When *n* > 10, the summation will be slightly more complicated as there will be more than one constant time operation when *n* > 10. There will still be *n* operations from the execution of the outer part of the loop that executes for each *n*. The inner loop will have one operation performed from *j* =1 to the current *i*.Again, under the assumption that each operation is executed in constant time, the summation would become:

For these particular values of *n* > 10, the time complexity of the code fragment will be O(*n*2) with *c0* = 2 and *n*0 = 11 (since this would only be the running time for *n* > 10) since:

Which means that:

Since we have two different run times that are dependent on *n*, we should select the larger of the two running times since:

and will thus account for both running times. So, **the time complexity for this snippet of code is O(*n*2).**