Common Complexity Scenarios

This lesson summarizes our discussion of complexity measures and includes some commonly used examples and handy formulae to help you with your interview.

We'll cover the following

List of Important Complexities

The following list shows some common loop statements and how much time they take to execute.

Simple for-loop with an increment of size 1

```
for (int x = 0; x < n; x++) {
    //statement(s) that take constant time
}
```

Running time Complexity = T(n) = $(2n+2)+cn = (2+c)n+2$. Dropping the leading constants $\Rightarrow n+2$. Dropping lower order terms $\Rightarrow O(n)$.

Explanation: Java for loop increments the value x by 1 in every iteration from 0 till n-1 ([0, 1, 2, ..., n-1]). So `n` is first 0, then 1, then 2, ..., then n-1. This means the loop increment statement `x++` runs a total of n times. The comparison statement `x < n ;` runs $n+1$ times. The initialization `x = 0;` runs once. Summing them up, we get a running time complexity of the for loop of $n+n+1+1 = 2n+2$. Now, the constant time statements in the loop itself each run n times. Supposing the

the constant time statements in the loop itself each run n times. Supposing the

statements inside the loop account for a constant running time of c in each iteration, they account for a total running time of cn throughout the loop's lifetime. Hence the running time complexity is $(2n+2)+cn$.

For-loop with increments of size k

```
for (int x = 0; x < n; x+=k) {
    //statement(s) that take constant time
}
```

Runing Time Complexity = $2+n(\frac{2+c}{k})$ = $O(n)$

Explanation: The initialization `x = 0;` runs once. Then, `x` gets incremented by `k` until it reaches `n`. In other words, `x` will be incremented to [$0, k, 2k, 3k, \cdots, (mk) < n$]. Hence, the incrementation part `x+=k` of the for loop takes $floor(\frac{n}{k})$ time. The comparison part of the for loop takes the same amount of time and one more iteration for the last comparison. So this loop takes $1+1+\frac{n}{k}+\frac{n}{k}=2+\frac{2n}{k}$ time. While the statements in the loop itself take $c \times \frac{n}{k}$ time. Hence in total, $2+\frac{2n}{k}+\frac{cn}{k}=2+n(\frac{2+c}{k})$ times, which eventually give us $O(n)$.

Simple nested For-loop

```
for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){
        //Statement(s) that take(s) constant time
    }
}
```

Running Time Complexity = $nm(2+c)+2+4n=O(nm)$

Explanation: The inner loop is a simple for loop that takes $(2m+2)+cm$ time and the outer loop *runs* it n times so it takes $n((2m+2)+cm)$time . Additionally, the initialization, increment and test for the outer loop take $2n+2$ time so in total, the running time is $n((2m+2)+cm)+2n+2=2nm+4n+cnm+2=nm(2+c)+4n+2$, which is $O(nm)$.

Nested For-loop with dependent variables

```
for (int i=0; i<n; i++){
    for (int j=0; j<i; j++){
        //Statement(s) that take(s) constant time
    }
}
```

Running Time Complexity = $O(n^2)$

Explanation: The outer loop runs n times and for each time the outer loop runs, the inner loop runs `i` times. So, the statements in the inner loop do not run at the first iteration of the outer loop since `i` is 0 then; they run *once* at the second iteration of the outer loop since `i` is equal to 1 at that point, then they run *twice*, then *thrice*, until `i` is $n-1$. So, they run a total of $c + 2c + 3c + \cdots + (n-1)c$ times = $c\left(\sum_{i=1}^{n-1} i\right) = c\frac{(n-1)((n-1)+1)}{2} = \frac{cn(n-1)}{2}$. The initialization of `j` in the inner for loop runs once in each iteration of the outer loop. So, this statement incurs a running time of n. In the first iteration of the outer for loop, the `j < i` statement runs once, in the second iteration it runs twice and so on. So, it incurs a total running time of $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. In each iteration of the outer loop, the `j++` statement runs one less time than the `j < i` statement, so it accounts for a running time of $0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$. So in total, the inner loop has a running time of $\frac{cn(n-1)}{2} + \frac{n(n+1)}{2} + \frac{n(n-1)}{2} + n$. The outer loop initialization, test and increment operations account for a running time of $2n + 2$. That means the entire script has a running time of $2n + 2 + \frac{cn(n-1)}{2} + n + \frac{n(n+1)}{2} + \frac{n(n-1)}{2}$ which is $O(n^2)$

Nested For-loop With Index Modification

```
for (int i=0; i<n; i++){
    i*=2;
    for (int j=0; j<i; j++){
        // Statement(s) that take(s) constant time
    }
}
```

Running Time Complexity = $O(n)$

Explanation: Notice that the outer loop index variable is modified in the loop's body. The first column in the following table shows the value of `i` immediately after entering the outer for loop. The second column shows the modified value of `i` after the `i*=2;` statement is run.

Outer Loop	Inner Loop
`i` = 0	`i` = 0*2 = **0**
`i` = 1	`i` = 1*2 = **2**
`i` = 3	`i` = 3*2 = **6**
$\cdots$	$\cdots$
`i` = $(n-1)$	`i` = $(n-1)\times 2$ = **2(n-1)**

A pattern is hard to decipher here. So, let's simplify things. In the outer loop, `i` is doubled and then incremented each time. If we ignore the increment part, we will be slightly over-estimating the number of iteration of the outer for loop. That is fine because we are looking for an upper bound on the worst-case running time (Big O).

If i keeps doubling, it will get from 1 to $n-1$ in roughly $log_2(n-1)$ steps. With this simplification, the outer loop index goes (approximately) $1, 2, 4, \cdots, 2^{log_2(n-1)}$. We've ignored the iteration with $i=0$, but it wouldn't affect the result in Big O. If you are interested, you can add 1 to all the steps in the following calculations. This sequence can also be written as $2^0, 2^1, 2^2, \cdots, 2^{log_2(n-1)}$. This series also gives the number of iterations of the inner for loop. Thus, the total number of iterations of the inner for loop is:

$$2^0+2^1+2^2+\cdots+2^{log_2(n-1)} = 2^{log_2(n-1)+1}-1 = 2^{log_2(n-1)}2-1 = 2(n-1)-1 = 2n-3$$

.

Therefore, the running time of the inner for loop is $2(2n-3)+2+c(2n-3)$ where c is the running time of the statements in the body of the inner loop. This simplifies to $2n(2+c)-3c-4$. The contribution from the initialization, test, and simplifies to $2n(2+c) - 3c - 4$. The contribution from the initialization, test, and

increment operations of the outer for loop is $2log_2(n-1)+2$. So, the total running time is $2n(2+c) - 3c - 4 + 2log_2(n-1)+2$. The term linear in n dominates the others, and the time complexity is $O(n)$.

Note that we could have done a rough approximation saying that the outer loop runs at most n times, the inner loop iterates at most n times each iteration of the outer for loop. That would lead us to conclude that the total running time is $O(n^2)$. Mathematically that is correct, but it isn't a tight bound.

Loops with log(n) time complexity

```
i = //constant
n = //constant
k = //constant
while (i < n){
    i*=k;
    // Statement(s) that take(s) constant time
}
```

Running Time Complexity = $\log_k(n)$ = $O(\log_k(n))$

Explanation: A loop statement that multiplies/divides the loop variable by a constant such as the above takes $\log_k(n)$ time because the loop runs that many times. Let's consider an example where n = 16, and k = 2:

i	Count
1	1
2	2
4	3
8	4
16	5

$$\log_k(n) = \log_2(16) = 4$$

Now that you have all the tools necessary to solve the time complexity problems let’s look at some exercises in the next few lessons.

Back

Useful Formulae

Next

Challenge 1: Big (O) of Nested Loop w...