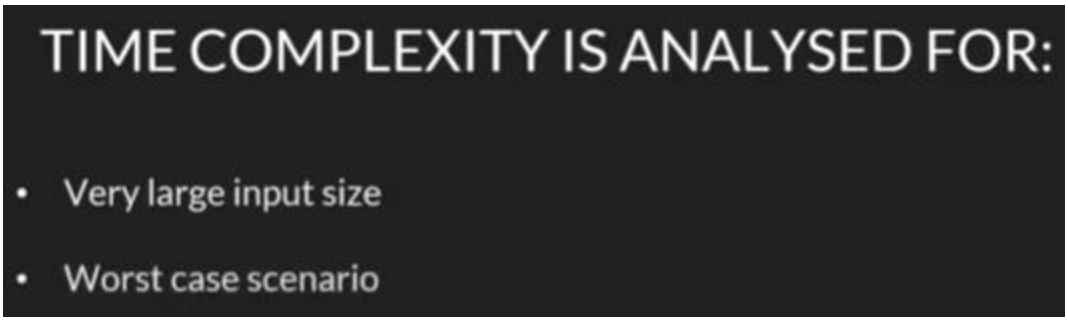


Let's say we have the following example:

$T(n) = 2n^2 + 3n + 1$

First, step is to drop lower order terms:

$T(n) = 2n^2 + \cancel{3n + 1}$

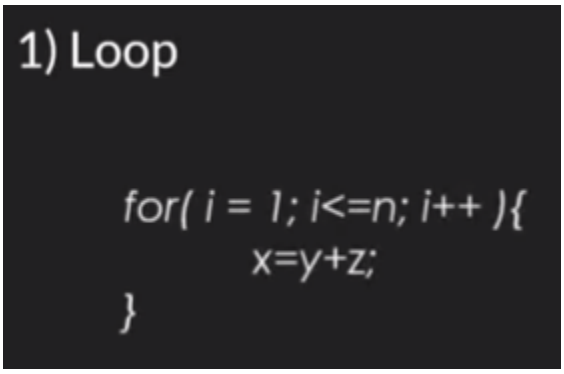
Next, we drop the constant multiplier:

$T(n) = \cancel{2}n^2 + \cancel{3n + 1}$

… so we get:

$= O(n^2)$

Now, let's examine the rules for finding time complexities for the following algorithms:



… the loop takes ***constant time*** to execute (**x = y+z**)

The time complexity will be ***n*** times the time taken for the algorithm to execute... the time is a constant, and let's call it ***c*** : 

… so the time complexity will be ***c times n*** : ***cn***

… in big O terms, we can neglect c and write the time complexity as ***O(n)***.

2) Nested Loop

```
for( i = 1; i<=n; i++ ){
        for( j = 1; j<=n; j++ ){
                x=y+z;
        }
}
```

… as you can see, we have 2 loops that are nested.

The constant time for this algorithm to execute is **x=y+z**.

The top line is the outer loop (***i***), which will run ***n*** times.

The second line is the inner loop (***j***), which will run ***n*** times for each run of the outer loop.

```
for( i = 1; i<=n; i++ ){ //n times
        for( j = 1; j<=n; j++ ){ //n times
                x=y+z; //constant time
        }
}
```

The total run time will be n * n * constant time, which is...

$= O(n^2)$

3) Sequential Statements

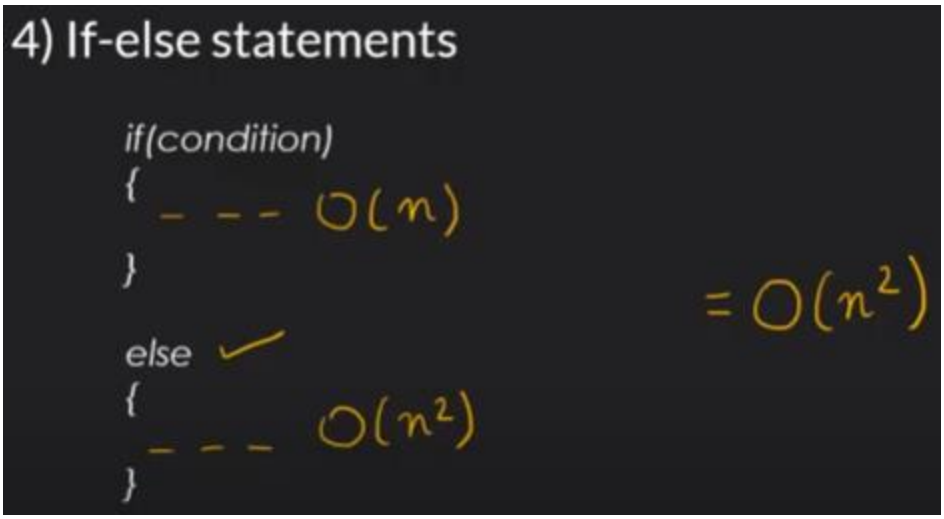
i) `a = a + b;` // constant time $= c_1$

ii)
```
for( i = 1; i<=n; i++ ){
        x = y + z;
}
```
$c_2 n$

iii)
```
for( j = 1; j<=n; j++ ){
        c = d + e;
}
```
$c_3 n$

$= c_1 + c_2 n + c_3 n$

$= O(n)$



Let's say in the if condition is a loop whose time complexity is ***O(n)***...

… and in the else statement is a nested for loop whose time complexity is ***O(n)²***...

… the else statement has a better time complexity, and thus the overall time complexity will be ***O(n)²***

Here is a chart of time complexities of various sorting algorithms:

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	θ(nk)	O(nk)

… it is highly recommended to memorize them, as they will come in handy.