How to determine Time Complexities

In general, how can you determine the running time of a piece of code? The answer is that it depends on what kinds of statements are used.

Sequence of statements statement 1; statement 2; ... statement k;	The total time is found by adding the times for all statements: total time = time(statement 1) + time(statement 2) + ... + time(statement k)

1. **O(1):** Time complexity of a function (or set of statements) is considered as O(1) if it doesn't contain loop, recursion and call to any other non-constant time function.
   In other words we can say, If each statement is "simple" (only involves basic operations) then the time for each statement is constant and the total time is also constant: O(1).

   // set of non-recursive and non-loop statements
   Example

```
void swap(int *x, int *y)
{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}
```

   A loop or recursion that runs a constant number of times is also considered as O(1).
   Example

```
// Here c is a constant
int i;
for (i = 1; i <= c; i++)
{
      // some O(1) expressions
 }
```

If-Then-Else	

if (expr) then
Block1 (sequence of statements)
else
Block2 (sequence of statements)
end if
	Here, either block 1 will execute, or block 2 will execute. Therefore, the worst-case time is the slower of the two possibilities:
	max(Time(Block1), Time(Block2))
	If block 1 takes O(1) and block 2 takes O(n), the if-then-else statement would be O(n).

LOOPS	
for i ← 1 to n sequence of statements endfor	The loop executes n times, so the sequence of statements also executes n times. If we assume the statements in loop are O(1), the total time for the for loop is N * O(1), which is O(N) overall.

2. **O(n):** Time Complexity of a loop is considered as O(n) if the loop variables is incremented / decremented by a constant amount. For example following functions have O(n) time complexity.

```
// Here c is a positive integer constant
int i ;
for (i = 1; i <= n; i += c)
{
    // some O(1) expressions
}

int i ;
for ( i = n; i > 0; i -= c)
{
    // some O(1) expressions
}
```

NESTED LOOPS	

for i ← 1 to m
  for j ← 1 to n
  sequence of statements
  endforj
endfori
	The outer loop executes m times. Every time the outer loop executes, the inner loop executes n times. As a result, the statements in the inner loop execute a total of m * n times. Thus, the complexity is O(m * n).

3. **O(n^k):** Time complexity of nested loops is equal to the number of times the innermost statement is executed. For example the following sample loops have O(n^2) time complexity

```
for (int i = 1; i <=n; i += c) {
    for (int j = 1; j <=n; j += c) {
      // some O(1) expressions
    }
}

for (int i = n; i > 0; i += c) {
    for (int j = i+1; j <=n; j += c) {
      // some O(1) expressions
}


for (i = 0; i < m; i++)
{
    for (j = 0; j < n; j++)
   {
        sequence of statements of O(1)
   }
}
```

The outer loop executes m times and inner loop executes n times so the time complexity is O(m*n)

```
for (i = 0; i < n; i++)
```

```
{
  for (j = 0; j < n; j++)
  {
    sequence of statements of O(1)
  }
}
```

Now the time complexity is $O(n^2)$

```
for (i = 0; i < n; i++)
{
     for (j = i+1; j < n; j++)
     {
          sequence of statements of O(1)
     }
}
```

Let us see how many iterations the inner loop has:

4. **O(log n)**: Time Complexity of a loop is considered as O(log n) if the loop variables is divided / multiplied by a constant amount.

```
for (int i = 1; i <=n; i *= c)
{
     // some O(1) expressions
}
for (int i = n; i > 0; i /= c)
{
     // some O(1) expressions
}
```

5. **O(log log n):** Time Complexity of a loop is considered as O(loglog n) if the loop variables is reduced / increased exponentially by a constant amount.

```
  // Here c is a constant greater than 1
  for (int i = 2; i <=n; i = pow(i, c))
  {
     // some O(1) expressions
  }

//Here fun is sqrt or cuberoot or any other constant root
 for (int i = n; i > 0; i = fun(i))
{
     // some O(1) expressions
}
```

6. When there are consecutive loops, we calculate time complexity as sum of time complexities of individual loops.

```
for (int i = 1; i <=m; i += c)
{
      // some O(1) expressions
}
```

```
for (int i = 1; i <=n; i += c)
{
      // some O(1) expressions
}
```
Time complexity of above code is O(m) + O(n) which is O(m+n)
If m == n, the time complexity becomes O(2n) which is O(n).