Iteration 1

Length of the array = n

Iteration 2

The array is divided in two

Length of the array =

Iteration 3

*The chosen half is divided in two*

Length of the array = = =

Iteration 4

*The chosen half is divided in two*

Length of the array = = =

\*We have found a pattern, now we can know the length of the array after the k iteration\*

After Iteration k

Length of the array =

\*After k divisions the length of the array is 1. Because we divide, divide and divide the array until we find (or not) the number\*

= 1

=

=

To get the time complexity (Big O) we need the value of k. So, we apply the binary logarithm on both sides to have the k alone on one side later.

=

We can apply the power rule property of logarithms, that way, we can extract the k from the logarithm.

|  |
| --- |
| \*Power rule\* |

=

Now, we can apply the identity rule to get rid of the

|  |
| --- |
| \*Identity rule\* |

=

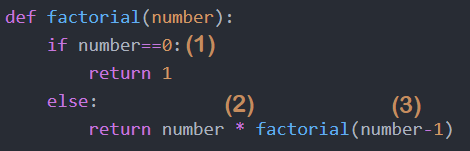
The complexity time of Binary Search is

O()

Recursive function It is a function that calls itself. It has a base case and a recursive case. Base case is when the function doesn't call itself and recursive case is when the function calls itself.

If we want the factorial of 0, it will just enter the base case and return 1. For any other number, it will have to go into recursion. Here is the example of factorial of 3

First, we compute how many operations there are in a basic call



1. If condition
2. Multiplication
3. Subtraction

factorial(0) will only enter the base case, so we can assume that it is constant. Therefore, factorial(0) = O(1) = 1 unit of time

factorial(n) will have 1if, 1 multiplication, 1 subtraction and time for factorial(n-1)

T(n) is the time complexity of the factorial recursive program.

\*Base case\*

T(0) = 1

\*Recursive cases\*

T(n) = T(n -1) +3 //First Recursion

= T(n - 2) + 6 //Second Recursion

= T(n - 3) + 9 //Third Recursion

\*We start to see a pattern\*

= T(n - k) + k\*3 //k Recursion

So, we know that T(0) = 1. This can help us to find the value of k and get the time complexity, but for that we need to convert to in terms of n.

If we need T(0), then we need to find the value of k for which n – k = 0, we pass k to the other side, n = k

T(n) = T(0) + 3n

= 1 + 3n

\*We remove the constants\*

= n

T(n) = O(n)