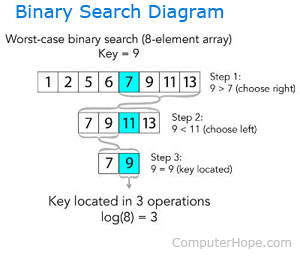
**Recursion Complexity**

The big-O runtime for a recursive function **is equivalent to the number of recursive function calls**. This value varies depending on the complexity of the algorithm of the recursive function. For example, a recursive function of input N that is **called N times** will have a runtime of **O(N**). On the other hand, a recursive function of input N that **calls itself twice per function** may have a runtime of **O(2^N).**

def countdown(value):  
  if value <= 0:  
    print("done")  
  else:  
    print(value)  
    countdown(value-1)  #recursive step

above function has a O(n) time complexity.

Example binary search



For every recursive call the input size gets halved.

N -> N/2 -> N/4 -> …...

Let T(n) be the recursive function

(using substitution method)

T(n) = T(n/2) + C (1)

T(n/2) = T(n/4) + C (2)

T(n/4) = T(n/8) + C (3)

Putting (2) on (1)

T(n) = T(n/4) + C + C (e1)

Putting (3) on (e1)

T(n) = T(n/8) + C + C + C

T(n) = T(n / 2^3) + 3C

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

For i th step

T(n) = T(n / 2^i) + iC

Let , 2 ^ i = n

T(n) = 1 + iC

log(2^i) = log(n)

i \* log(2) = log(n) | (log2 = 1)

i = log(n)

T(n) = 1 + log(n).C

T(n) = O(log(n))