RECURSION

1. What is Recursion?

 The process of having a method continually call itself until a defined point of termination.

1. Requirements:

* One or more base cases without recursive call (smallest problem size)
* One or more recursive cases in which the algorithm is defined in term of itself
* The recursive cases MUST eventually lead to a base case

3.Why use recursion?

• Recursion is often more elegant – Call stack handles some complexities

• Recursion is sometimes easier to understand – Some problems easier to describe with recursion

• Recursion simpler when multiple calls – Iterative implementation too complex

4. Examples:

* Suppose we want to calculate x \* y where y is a positive integer but what if our computer do not have the multiplication operator.



Recursively:



5.Exercise on Recursion:

* Write a recursive function to calculate the factorial.
* Write a recursive function to calculate power(x,n).

6. Solution:





7. Finding the running Time complexity for these two problems:

Factorial:

* The key instruction is n \* factorial(n - 1);
* If it is an iterative version, we simply count the number of times the above statement is executed.
* Note that every time the key instruction is executed, it make a recursive call.
* It may be simpler to simply count the number of recursive calls.
* For this analysis, we need to know the total number of recursive calls base on the problem size n.
* Get the formula that represents the number of recursive calls (e.g., t(n))
* Find the Big-Oh of t(n)
  + When n = 0, no recursive call (simply return 1)
  + When n = 1, total of one recursive call
    - Call factorial(0)
    - Note that factorial(0) does not make any recursive calls.
  + When n = 2, total of two recursive calls
    - Call factorial(1)
    - Note that factorial(1) makes one recursive call.
  + When n = 3, total of three recursive calls
    - Call factorial(2)
    - Note that factorial(2) makes two recursive calls.
  + Let t(n) be the total number of recursive calls of
  + factorial(n), we can observe that
    - t(0) = 0 and
    - t(n) = 1 + t(n - 1) → ( Recursive Relation)
      * 1 → call factorial(n - 1)
      * t(n -1) → the total number of recursive calls of factorial(n-1)
  + It appears that t(n) = n for n >=0.
* We need to prove that t(n) = n for n >=0.
  + Base Case: n = 0

If t(n) = n, t(0) must be 0. We already see that when n = 0, no recursive call.

* + Induction Step: Assume that t(n) = n is true for any 0<= n <= k for some positive integer k. We need to show that t(k + 1) = k + 1.
  + From our recursion relation, t(n) = 1 + t(n - 1). Thus,
    - t(k + 1) = 1 + t(k + 1 - 1) = 1 + t(k)
  + From our assumption, we assume that t(n) = n for any 0 <= n <= k. Thus t(k) = k. Therefore,
    - t(k + 1) = 1 + t(k) = 1 + k
  + Since t(n) = n, our factorial(n) is O(N).

Power:

* When n = 0, no recursive call
* When n = 1, one recursive call (1 \* power(x,0))
* When n = 2, two recursive calls (2 \* power(x,1))
* When n = 3, three recursive calls (3 \* power(x,2))
* We can observe that
  + t(0) = 0 and
  + t(n) = 1 + t(n - 1)
* Similar to our factorial(n), the number of recursive calls of power(x,n) is t(n) = n.
* Thus, our power(x,n) is O(N).

**FASTER VERSION OF POWER:**





