Analysis of Recursive Algorithms

Procedure for Recursive Algorithm

1. Decide on parameter n indicating input size
2. Identify algorithm's basic operation
3. Determine worst, best, and average case for input of size n
4. Set up a recurrence relation and the initial conditions (IC) for T(n) – the no. of times the basic operation will be executed for an input of size n
5. Solve the recurrence relation using suitable method and find the order of growth

Example: Factorial of a number

$n!$ = 1•2•3...n and 0! = 1 (called initial/base case)
So the recursive definition is n! = n*(n-1)!

Algorithm $F(n)$
    if n = 0 **then** return 1 // base case
    else F(n-1)•n // recursive call

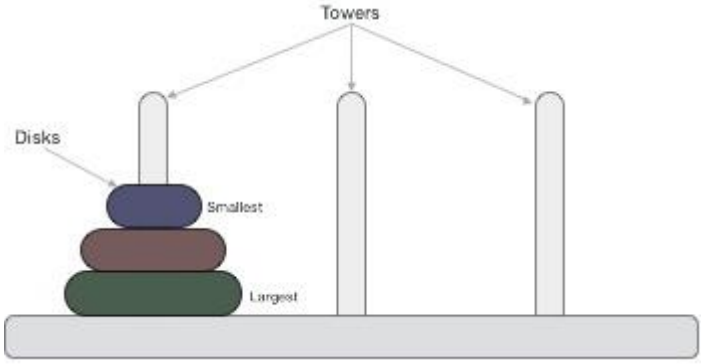
Analysis
Basic operation: multiplication during the recursive call
The number of multiplications is given by
    $M(n) = M(n-1) + 1$ (Recursive step)
with the initial condition
    $M(0) = 0$ (Basic step)
There are no multiplications

Solve by the method of *backward substitutions*
    $M(n) = M(n-1) + 1$
    $= [M(n-2) + 1] + 1 = M(n-2) + 2$ substituted $M(n-2)$ for $M(n-1)$
    $= [M(n-3) + 1] + 2 = M(n-3) + 3$ substituted $M(n-3)$ for $M(n-2)$
    ……..
    ……..
    $= M(n-k) + k$ *(after k step)*
    Put $k = n$, we get
    $M(n) = M(0) + n = 0 + n$

Therefore, $M(n) = \Theta(n)$

Example: Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −



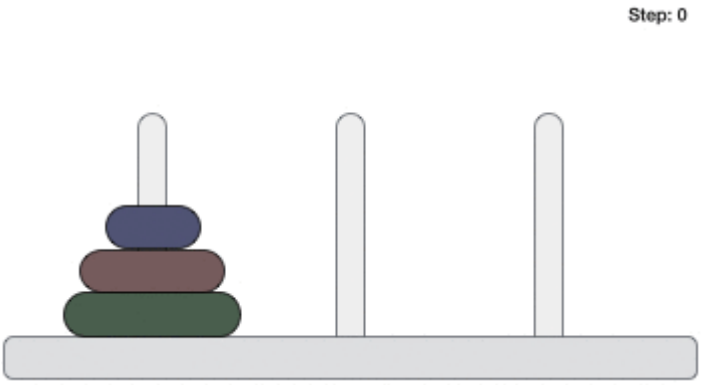
These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Following is an animated representation of solving a Tower of Hanoi puzzle with three disks.



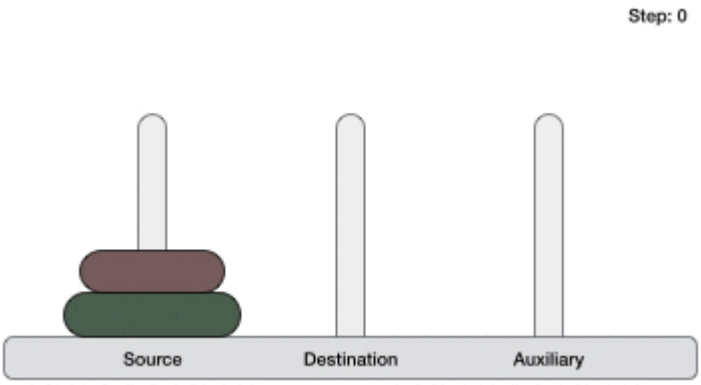
Tower of Hanoi puzzle with n disks can be solved in minimum $\mathbf{2^n-1}$ steps. This presentation shows that a puzzle with 3 disks has taken $\mathbf{2^3 - 1 = 7}$ steps.

Algorithm

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, **source**, **destination** and **aux** (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks −

- First, we move the smaller (top) disk to aux peg.
- Then, we move the larger (bottom) disk to destination peg.
- And finally, we move the smaller disk from aux to destination peg.



So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (n^{th} disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk **n** from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.

The steps to follow are −

Step 1 − Move n-1 disks from **source** to **aux**

Step 2 − Move n^{th} disk from **source** to **dest**

Step 3 − Move n-1 disks from **aux** to **dest**

Recursive algorithm for Tower of Hanoi problem is

```
Algorithm Hanoi(n, source, dest, aux)
   IF n = 1, THEN
    move disk from source to dest
  ELSE
    Hanoi(n - 1, source, aux, dest)    // Step 1
    move disk from source to dest      // Step 2
    Hanoi(n - 1, aux, dest, source)    // Step 3
  END IF
```

Analysis

1. Problem size is *n*, the number of discs
2. The basic operation is moving a disc from one peg to another
3. There is no worst or best case
4. Recursive relation for moving n discs

   $M(n) = M(n\text{-}1) + 1 + M(n\text{-}1) = 2M(n\text{-}1) + 1$

   IC: $M(1) = 1$
5. Solve using backward substitution

   $M(n) = 2M(n\text{-}1) + 1$

   $= 2[2M(n\text{-}2) + 1] + 1 = 2^2M(n\text{-}2) + 2 + 1$

   $= 2^2[2M(n\text{-}3) + 1] + 2 + 1 = 2^3M(n\text{-}3) + 2^2 + 2 + 1$

   ……

   ……

$M(n) = 2^i M(n-i) + \sum_{j=0}^{i-1} 2^j = 2^i M(n-i) + 2^i - 1$ (after i^{th} step)

Put $i = n-1$, we get
$M(n) = 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1 = 2^{n-1} M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1$

Therefore, $M(n) = \Theta(2^n)$

Example: Fibonacci Numbers Sequence

Fibonacci Sequence: 0, 1, 1 2, 3, 5, 8, 13, 21, 34, ...
Fibonacci (1202) proposed for the growth of rabbits
Can be defined by the simple recurrence

$F(n) = F(n-1) + F(n-2)$ for $n > 1$

With the initial conditions

$F(0) = 0$ and $F(1) = 1$

Homogenous second-order linear recurrence with constant coefficients

$ax(n) + bx(n-1) + cx(n-2) = 0$

Homogenous because the recurrence equals zero.
Why second-order linear? Substitute the proposed solution $x(n) = r^n$

$a r^n + b r^{n-1} + c r^{n-2} = 0$

divide by r^n

$a + b r + c r^2 = 0$, *characteristic equation* is a second order polynomial.

The real roots are solutions

$x(n) = \alpha r_1^n + \beta r_2^n$

α and β are the constants and need to be determined from the initial condition

Apply to Fibonacci Recursion

$F(n) - F(n-1) - F(n-2) = 0$, homogenous second order with constant coefficients

The characteristic equation is given by

$r^2 - r - 1 = 0$

$r_{1,2} = (1 \pm \sqrt{5})/2 = \varphi$ or φ' { $\varphi = (1+\sqrt{5})/2$ and $\varphi' = (1-\sqrt{5})/2$}

The general form of the solution will be

$F(n) = \alpha\varphi^n + \beta\varphi'^n$ where α and β are unknows

Using the initial conditions

$\alpha + \beta = 0$

$\varphi\alpha + \varphi'\beta = 1$

By solving, we get

$\alpha = 1/\sqrt{5}$ and $\beta = -1/\sqrt{5}$

So

$F(n) = 1/\sqrt{5}\ (\varphi^n - \varphi'^n) = \varphi^n/\sqrt{5}$ rounded to the nearest integer

Example: Recursive Algorithm for Fibonacci Numbers

Algorithm $F(n)$

if $n \leq 1$ **then return** n

else return $F(n\text{-}1) + F(n\text{-}2)$

1. Problem size is n, the sequence number for the Fibonacci number
2. Basic operation is the sum in recursive call
3. No difference between worst and best case
4. Recurrence relation

$A(n) = A(n\text{-}1) + A(n\text{-}2) + 1$

IC: $A(0) = A(1) = 0$

or

$A(n) - A(n\text{-}1) - A(n\text{-}2) = 1$, **non-homogeneous recurrences** because of 1

In general solution to the non-homogeneous problem is equal to the sum of solution to homogenous problem plus solution only to the non-homogeneous part. The undetermined coefficients of the solution for the homogenous problem are used to satisfy the IC.

In this case $A(n) = B(n) + I(n)$, where
$A(n)$ is solution to complete non-homogeneous problem
$B(n)$ is solution to homogeneous problem
$I(n)$ solution to only the non-homogeneous part of the problem

We guess at $I(n)$ and then determine the new IC for the homogenous problem for $B(n)$
For this problem the correct guess is $I(n) = 1$
Substitute $A(n) = B(n) - 1$ into the recursion, we get
$B(n) - B(n\text{-}1) - B(n\text{-}2) = 0$

with initial conditions $B(0) = B(1) = 1$

The same as the relation for $F(n)$ with different initial condition
We do not really need the exact solution;
Thus

$A(n) = B(n) - 1 = F(n+1) - 1 = \Theta(\varphi^n)$, exponential

Iterative algorithm for Fibonacci numbers

Algorithm $Fib(n)$

$F[1] \leftarrow 0$; $F[1] \leftarrow 1$

for i $\leftarrow 2$ **to** n **do**

$F[\text{i}] \leftarrow F[i\text{-}1] + F[i\text{-}2]$

return $F[n]$

Order of growth is $\Theta(n)$.