Text Book:
Introduction to the Design and Analysis of Algorithms
Author: Anany Levitin
2nd Edition

Mathematical Analysis of Recursive algorithms

General Plan for Analysing the Time Efficiency of Non-recursive Algorithms

1. Decide on a *parameter* (or parameters) indicating an input's size.
2. Identify the algorithm's *basic operation* (The operation that consumes maximum amount of execution time).
3. Check whether the *number of times the basic operation is executed* depends only on the size of an input. If the number of times the basic operation gets executed varies with specific instances (inputs), we need to carry out Best, Worst and Average case analysis
4. *Set up a recurrence relation*, with an appropriate initial condition, for the number of times the basic operation is executed.
5. Solve the recurrence to determine time complexity, establish **order of growth** of its solution

Methods to solve recurrences

- Substitution Method
  - Mathematical Induction
  - Backward substitution
- Recursion Tree Method
- Master Method (Decrease by constant factor recurrences)

EXAMPLE 1:

Find n!

n ! = 1 * 2 * … *(n-1) * n for $n \geq 1$ and 0! = 1

Recursive definition of n!:

$F(n) = F(n-1) * n$ for $n \geq 1$

$F(0) = 1$

ALGORITHM F(n)
//Computes n! recursively
//Input: A nonnegative integer n
//Output: The value of n!
if n = 0
  return 1
else
  return F(n − 1) * n

Algorithm analysis

Input size: n
Basic operation: Multiplication
Best/Worst/Average case: number of multiplications depend only on n so no best/worst/average case analysis
Recurrence relation and initial condition for number of multiplications required to compute n! is given as

M(n) = M(n − 1) + 1 for n > 0,
M(0) = 0 for n = 0.

Method of backward substitutions
M(n) = M(n − 1) + 1 substitute M(n − 1) = M(n − 2) + 1
= [M(n − 2) + 1]+ 1
= M(n − 2) + 2 substitute M(n − 2) = M(n − 3) + 1
= [M(n − 3) + 1]+ 2
= M(n − 3) + 3
…
= M(n − i) + i
M(0)=0 So substitute i=n
= M(n − n) + n
= n.
Therefore M(n)=n
⇒T(n) ∈ Θ (n)

EXAMPLE 2:

Tower of Hanoi

ALGORITHM TOH(n, A, C, B)
//Move disks from source to destination recursively
//Input: *n disks and* 3 pegs A, B, and C
//Output: Disks moved to destination as in the source order.

```
    if  n=0
        return
  else
      Move top n-1 disks from A to B using C
      TOH(n - 1, A, B, C)
      Move 1 disk from A to C
      Move top n-1 disks from B to C using A
      TOH(n - 1, B, C, A)
```

Algorithm analysis

M(n) = **2M(n-1) + 1** for n > 0 and M(0)=0
= **$2^n - 1 \in \Theta(2^n)$**
$\Rightarrow T(n) \in \Theta(2^n)$

EXAMPLE 3

Determine number of binary digits in the binary representation of a positive decimal integer

ALGORITHM BinRec(n)
//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation

```
    if n = 1
       return 1
    else
       return BinRec(floor(n/2))+ 1
```

Algorithm analysis

Input size=n
Basic operation: Addition
Number of additions depend only on the size of input n so no separate analysis is required for best, worst and average case

Recurrence Relation for number of additions

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{for } k > 0,$$
$$A(2^0) = 0.$$

$$\begin{aligned}
A(2^k) &= A(2^{k-1}) + 1 && \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1\\
&= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 && \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1\\
&= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 && \cdots\\
&\cdots\\
&= A(2^{k-i}) + i\\
&\cdots\\
&= A(2^{k-k}) + k.
\end{aligned}$$

$$A(n) = \log_2 n \in \Theta(\log n).$$