DAA

UNIT-I. 1

Worst-case, Best-case & Average-case efficiencies

SequentialSearch (A[0....n-1], K)

// Searches for a given values in a given array by sequential search

// Input: An array A[0..n-1], and a search key K

// output: Returns the index of the first element of A that matches K or -1 if there are no matching elements

i = 0

while i < n and A[i] ≠ K do

  i = i+1

if i < n return i

else return -1

basic operation: comparision

worst case

i) no matching elements or

ii) matching element present in the last location of the list

Algorithm performs maximum number of key comparisions

$$\boxed{C_{worst}(n) = n}$$

Best-case

If the search key is present in the first location of the list, algorithm performs only one comparision.

$$\boxed{C_{best}(n) = 1}$$

Average-case

2 assumptions

1) $P \rightarrow$ probability of successful search $(0 \leq P \leq 1)$
2) the probability of the first match occuring in the ith position of the list is the same for every i.

$$C_{avg}(n) = \left[1 \cdot \frac{P}{n} + 2 \cdot \frac{P}{n} + \dots n \cdot \frac{P}{n}\right] + n(1-P)$$

$$= \frac{P}{n}[1+2+\dots n] + n[1-P]$$

$$= \frac{P}{\cancel{n}} \frac{\cancel{n}(n+1)}{2} + n[1-P] = \frac{P(n+1)}{2} + n[1-P]$$

$$\boxed{C_{avg}(n) = \frac{P(n+1)}{2} + n(1-P)}$$

case i) $P=1$, successful search

$$\boxed{C_{avg}(n) = \frac{n+1}{2}}$$

ii) $P=0$, unsuccessful search

$$\boxed{C_{avg}(n) = n}$$

Asymptotic notations

$g(n) \rightarrow$ order of growth

Ex: 1. $t(n) = 10n^2 + 5n = O(n^2)$

$= \max(n^2, n) = O(n^2)$

2. $t(n) = 10 \log n + 5 = O(\log n)$

3. $t(n) = 5 \log n + n.\log n + 2^n$

$= \max(\log n, n.\log n, 2^n)$

$= O(2^n)$.

4. $t(n) = 10 \log n + 5n = O(n)$

Big-oh

O- notation

A function $t(n)$ is said to be in $O(g(n))$ denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

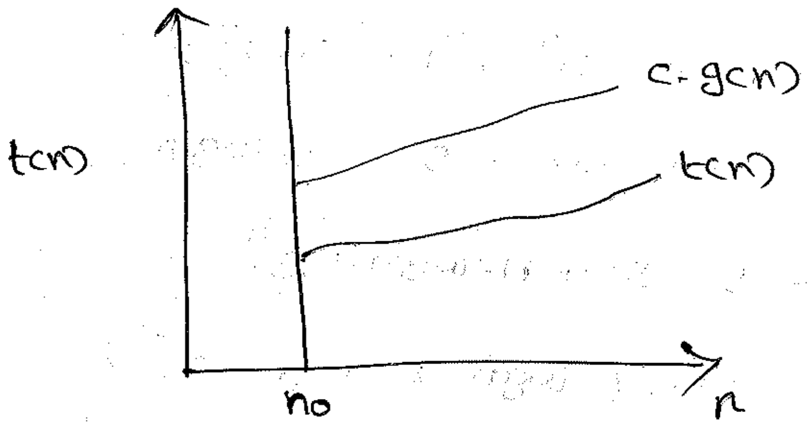
$$t(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 .$$

Ex :

$$100n + 5 \in O(n^2)$$

$$100n + 5 \leq 100n + n \ (n \geq 5) = 101n \leq 101 \cdot n^2$$

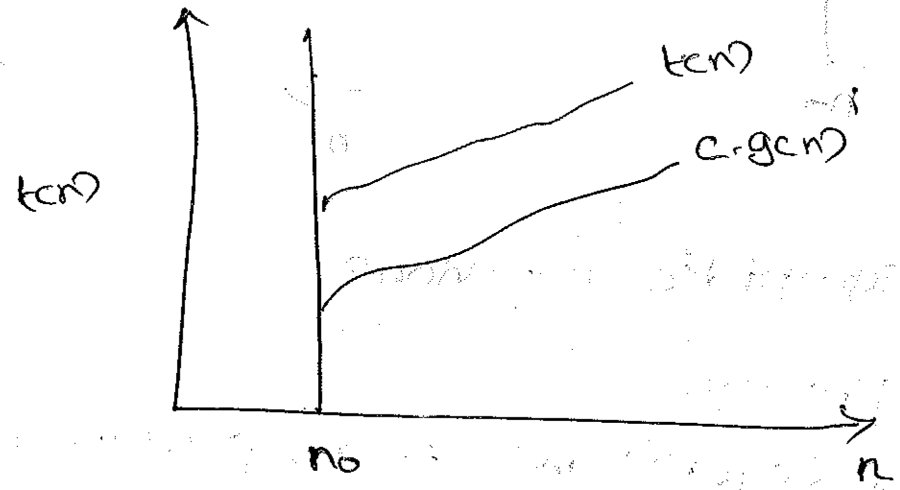
$$c = 101, n_0 = 5 .$$



Ω-notation Big-omega

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive ~~the~~ constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some non-negative integer no such that

$$t(n) \geq c \cdot g(n) \text{ for all } n \geq n_0.$$



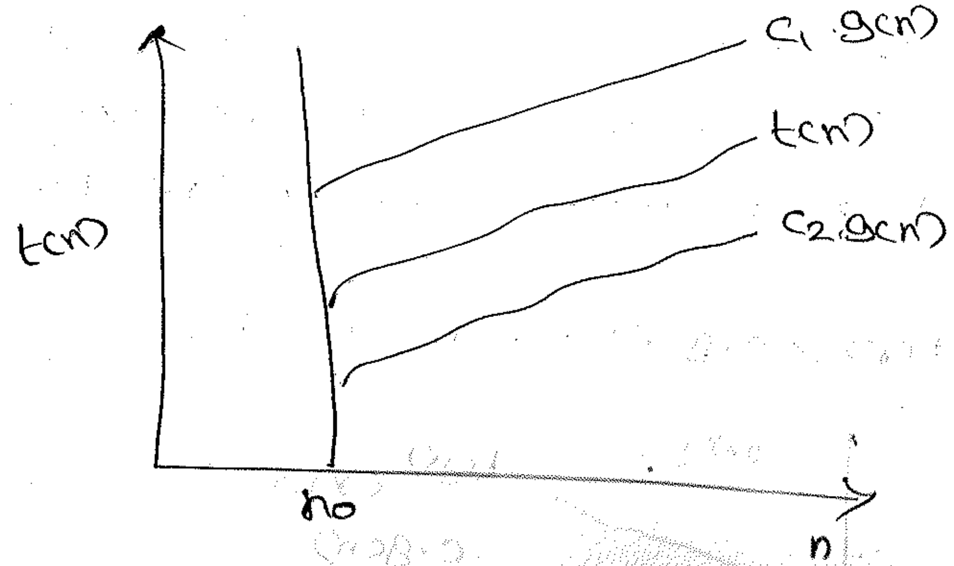
Ex: $n^3 \in \Omega(n^2)$,

$n^3 \geq n^2, \; n \geq 0$,

Θ notation. Big theta

A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n, i.e., if there exist some positive constant c_1 and c_2 and some non-negative integer no such that

$c_2\, g(n) \leq t(n) \leq c_1\, g(n)$ for all $n \geq n_0$.



properties of asymptotic notations

Sum function property

1. If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$,

then

$$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$$

<u>proof</u> $\rightarrow$ $t_1(n) \in O(g_1(n))$

$t_1(n) \leq c_1 . g_1(n)$, for all $n \geq n_1$.

$t_2(n) \in O(g_2(n))$,

$t_2(n) \leq c_2 . g_2(n)$, for all $n \geq n_2$.

$t_1(n) + t_2(n) \leq c_1\, g_1(n) + c_2\, g_2(n)$

$c_3 = \max\{c_1, c_2\}$

$n \geq \max\{n_1, n_2\}$

$$\le c_3 g_1(n) + c_3 g_2(n)$$

$$\le c_3 2[g_1(n) + g_2(n)]$$

$$\le c_3 \cdot 2 \max\{g_1(n), g_2(n)\}$$

$$t_1 + t_2 \in O(\max\{g_1(n), g_2(n)\})$$

2. product of two functions property

If $t_1(n) = O(g_1(n))$ and $t_2(n) = O(g_2(n))$

then $t_1(n) \times t_2(n) = O(g_1(n) \times g_2(n))$

proof:

$$t_1(n) \le c_1 g_1(n)$$

$$t_2(n) \le c_2 g_2(n)$$

$$t_1(n) \times t_2(n) \le c_1 g_1(n) \times c_2 g_2(n)$$

$$\le c_3 (g_1(n) \times g_2(n))$$

$$t_1(n) \times t_2(n) = O(g_1(n) \times g_2(n))$$

3. Transitive property:

If $t(n) = O(g(n))$ and $g(n) = O(h(n))$

then $t(n) = O(h(n))$

4. Reflexivity

$$t(n) = O(t(n))$$

5. Transpose

computing time

n	n^2	$\log n$
5	25	2
10	100	3

order of growth

$t(n) = 10n + 5n^2 = O(n^2)$.

$t(n) = n$ $\quad$ $t(n) = n^2$

Basic asymptotic efficiency classes

class	name
1	constant
$\log n$	logarithmic
n	linear
$n.\log n$	n - log - n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	Factorial

Mathematical analysis of non-recursive algorithms

1. pbm to find the largest element in a list of n numbers.

```
MaxElement (A[0..n-1])
 maxval = A[0]
 for i = 1 to n-1 do
     if A[i] > maxval
        maxval = A[i]
 return maxval.
```

$C(n)$ → number of times comparision is executed

n → input size   up-lo-1 $n = 4$

A

0	5
1	10
	12
n-1 3	7

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1-1+1 = \underline{\underline{n-1}}$$

$$C(n) = n-1 = O(n)$$

$$\boxed{C(n) = O(n)}$$

$$\boxed{C(n) = \sum_{i=1}^{n} 1 = ul - ll + 1} = n - \cancel{1} + \cancel{1}$$

$$= n$$

General plan

1. decide input size
2. identify basic operation
3. find number of times basic operation is executed
4. Set up a sum expressing the number of times the basic operation is executed
5. Find order of growth.

2. Element uniqueness problem

→ to check whether all the elements in a given array are distinct.

```
UniqueElements(A[0...n-1])
for i=0 to n-2 do
    for j=i+1 to n-1 do
        if A[i]= A[j] return false
return true
```

$i=0$ to $n-2$

$j=i+1$ to $n-1$

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

		A
	0	5
	1	10
	2	12
	3	18
$n-2$	4	2
$n-1$	5	16

$$= \sum_{i=0}^{n-2} (n-1-(i+1)+1)$$

$$= \sum_{i=0}^{n-2} (n-1-i\cancel{-1}\cancel{+1}) = \sum_{i=0}^{n-2} (n-1-i)$$

$$1+2+3+\dots n = \frac{n(n+1)}{2}$$

$$= \frac{n-2\,(n-2+1)}{2}$$

$$= \frac{(n-2)(n-1)}{2}$$

$$= \sum_{i=0}^{n-2}(n-1) - \sum_{i=0}^{n-2} i$$

$$= n-1 \sum_{i=0}^{n-2} 1 - \left(0+1+\dots (n-2)\right)$$

$$= (n-1)(n-2-0+1) - \frac{(n-1)(n-2)}{2}$$

$$= n-1 \left[(n-1) - \frac{(n-2)}{2} \right] = (n-1)\left[\frac{2n\cancel{-2}-n\cancel{+2}}{2}\right]$$

$$= \frac{n(n-1)}{2}$$

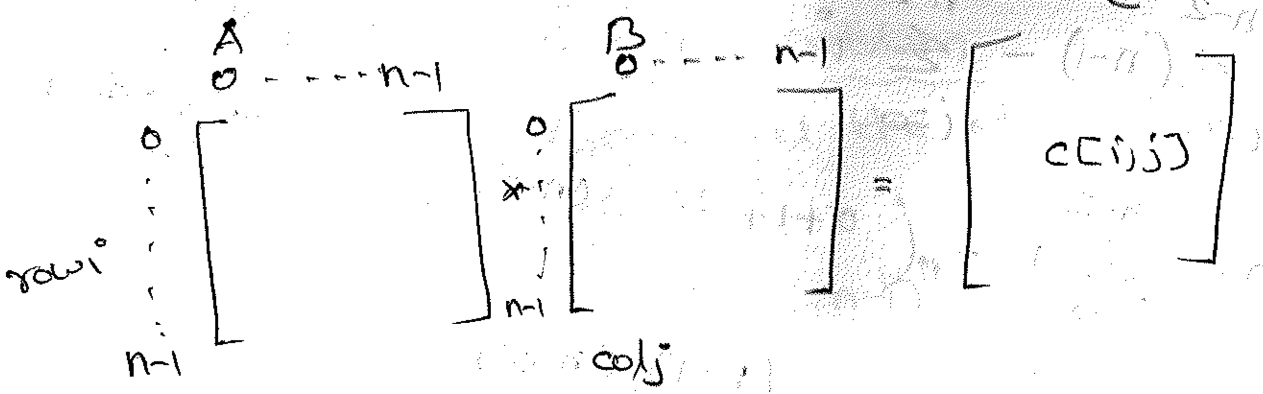
$$\boxed{C_{worst}(n) = \frac{n(n-1)}{2}}$$

3. Given two n by n matrices A and B, Find C = AB

MatrixMultiplication(A[0...n-1, 0...n-1], B[0...n-1, 0...n-1])

for i = 0 to n-1 do
    for j = 0 to n-1 do
        c[i, j] = 0
        for k = 0 to n-1 do
            c[i, j] = c[i, j] + A[i, k] * B[k, j]

return C



Basic operation: multiplication

Algorithm performs only one multiplication for every value of i, j and k.

Total number of multiplications M(n) is expressed by the following triple sum:

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [n-1-0+1] = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n =$$

$$= n \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1$$

$$= n \sum_{i=0}^{n-1} [n-1-0+1]$$

$$= n^2 \sum_{i=0}^{n-1} 1 = n^2 [n-1-0+1] = n^3$$

$$\boxed{M(n) = n^3 = O(n^3)}$$

$$T(n) = c_m M(n) + c_a A(n)$$

$$= c_m n^3 + c_a n^3$$

$$\boxed{T(n) = (c_m + c_a) n^3}$$

4) Find the number of binary digits in the binary representation of a positive decimal integer.

Binary (n)

count = 1

while n > 1 do

count = count + 1

$n \leftarrow \lfloor n/2 \rfloor$

return count

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = \log_2 2^k = \boxed{k = \log n}$

basic operation : comparision.

n = 8

8 > 1

4 > 1

2 > 1

1 > 1

$\frac{8}{2^3} = 1$

$\frac{4}{2^2} = 1$

$\frac{8}{2^3} = 1$

① n = ~~6~~ 2 = 10

c = 1 – $\boxed{2}$

n = 1

n = 8

1000

② n = 4 = 100

c = 1

c = 2

n = 2

$\boxed{c = 3}$

③ n = 8 ⇒ 1000

c = ~~1~~ 2

n = 4

c = 3

n = 2

$\boxed{c = 4}$ n = 1

c = 2

n = 4

c = 3

n = 2

$\boxed{c = 4}$

log n times the loop is repeated.

The number of times the comparision (n > 1) will be executed is larger than the number of repetitions of the loop's body by exactly 1.

$$\boxed{c(n) = \log_2 n + 1 = O(\log n)}$$

$n = 8$

$\underline{8>1,\ 4>1,\ 2>1},\ 1>1$, 4 comparisions.

le le le ×

$\frac{8}{2^{3}} = 1.$ loop is executed 3 times.

$\frac{8}{2} \cdot \frac{4}{2} \cdot \frac{2}{2}$

$\log 8 = \log_2 2^3 = 3.$

———— × ————

Mathematical Analysis of recursive algorithm

1. compute factorial $n!$

$n! = 1 \ldots\ldots (n-1) \cdot n = (n-1)! \cdot n, \ n \geq 1$

F(n)

if $n = 0$ return 1

else return $F(n-1) \times n$.

$F(n) = F(n-1) \cdot n, \ n > 0$

$M(n) \Rightarrow$ no. of multiplication

basic operation multiplication.

$M(n) = M(n-1) + 1, \ n > 0.$

$M(0) = 0$, if $n = 0$ no multiplication.

$F(0) = 1, \ 0! = 1.$

Method of backward substitution method,

$M(n) = M(n-1) + 1 \rightarrow ①$

$M(n-1) = M(n-2) + 1 \rightarrow ②$

$M(n-2) = M(n-3) + 1 \rightarrow ③$

sub ③ in ②

$M(n-1) = [M(n-3)+1] + 1 = M(n-3) + 2 \rightarrow ④$

Sub ④ in ①

$M(n) = [M(n-3) + 2] + 1$

$\boxed{5! = 5 \times 4 \times 3 \times 2 \times 1 = 5 \text{ multiplication}}$

$M(n) = M(n-3) + 3$

In general

$M(n) = M(n-i) + i$

sub $i = n$

$M(n) = M(n-n) + n$

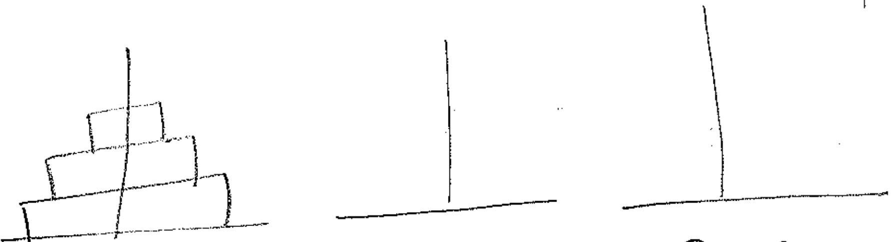
$M(n) = M(0) + n \quad [M(0) = 0]$

$\boxed{M(n) = n}$

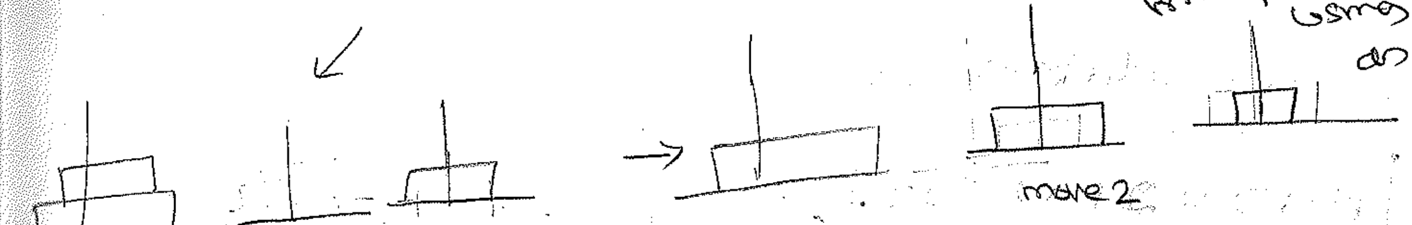
$\boxed{M(n) = O(n)}$

2. Tower of Hanoi puzzle.

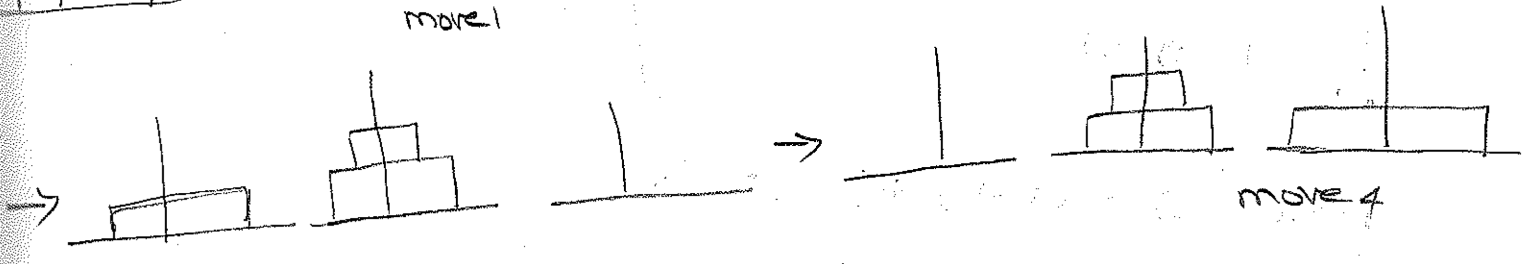
2. only one disk can be moved at a time
3. Smaller placed on larger one, not vice versa.
1. Move all disk from peg1 to peg3 using peg2 as intermediate



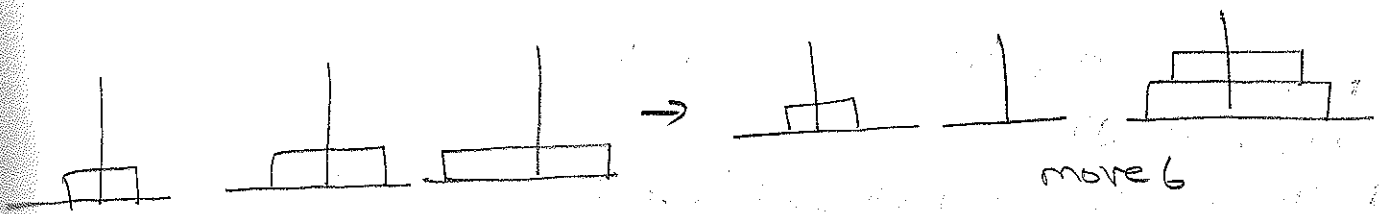
peg1 peg2 peg3



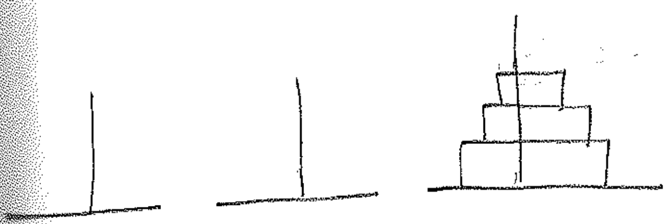
move 1 move 2



move 3 move 4



moves 5 move 6



move 7

no. of disk = 3.

$$\boxed{\text{no. of moves} = 2^n - 1} = 2^3 - 1 = 7$$

$M(n) \Rightarrow$ total number of moves.

$$M(n) = M(n-1) + n + M(n-1) \quad , n > 1$$

Initial condition

$$M(1) = 1.$$

Recurrence relation is

$$\boxed{\begin{aligned} M(n) &= 2M(n-1)+1 \quad , n>1 \\ M(1) &= 1 \end{aligned}}$$

$$M(n) = 2M(n-1) + 1 \rightarrow (1)$$

$$M(n-1) = 2M(n-2) + 1 \quad - (2)$$

$$M(n-2) = 2M(n-3) + 1 \rightarrow (3)$$

Sub (3) in (2)

$$M(n-1) = 2[2M(n-3)+1] + 1$$

$$M(n-1) = 2^2 M(n-3) + 2 + 1 \rightarrow (4)$$

Sub (4) in (1)

$$M(n) = 2[2^2 M(n-3) + 2 + 1] + 1$$

$$= 2^3 M(n-3) + 2^2 + 2 + 1$$

Next

$$M(n) = 2^4 M(n-4) + 2^3 + 2^2 + 2^1 + 1$$

In general substitute i

$$M(n) = 2^{i} M(n-i) + 2^{i-1} + 2^{i-2} + \dots 2^{1} + 2^{0}$$

$$\sum_{i=0}^{n-1} 2^{i} = 2^{n} - 1$$

$$M(n) = 2^{i} M(n-i) + 2^{i} - 1$$

Sub $i = n-1$

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

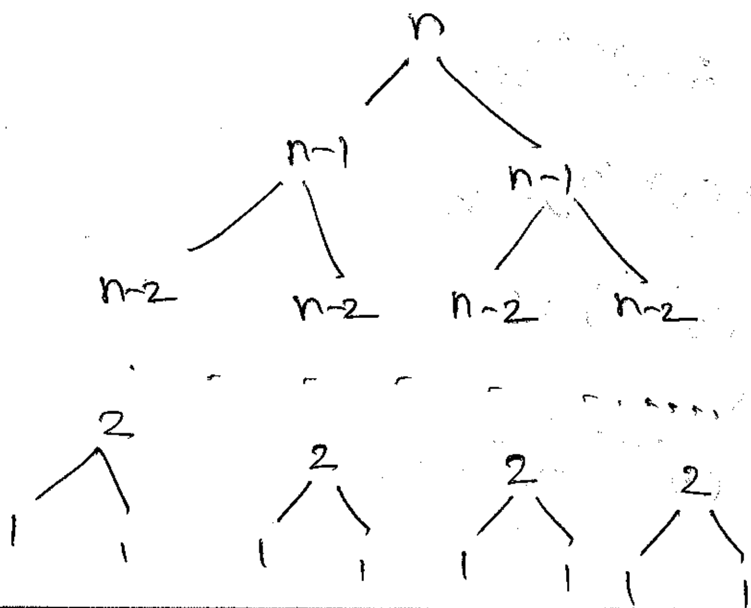
$$= 2^{n-1} M(1) + 2^{n-1} - 1 \quad [M(1) = 1]$$

$$= 2^{1} \cdot 2^{n-1} - 1$$

$$= 2^{n} - 1$$

$$\boxed{M(n) = 2^{n} - 1} \qquad \boxed{M(n) = O(2^{n})}$$

Tree of recursive calls



3. Recursive algorithm to find number of binary digits in the binary representation of a decimal number.

BinRec(n)

if n = 1 return 1

else return BinRec(⌊n/2⌋ +1)

basic operation: addition

$$A(n) = A(\lfloor n/2 \rfloor) + 1, \quad A(1) = 0$$

if n = 1, no addition.

$$\boxed{n = 2^k, \quad k = \log n}$$

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

$$A(2^{k-1}) = A(2^{k-2}) + 1$$

$$A(2^{k-2}) = A(2^{k-3}) + 1$$

$$A(2^{k-1}) = A(2^{k-3}) + 2$$

$$A(2^k) = A(2^{k-3}) + 3$$

In general

$$A(2^k) = A(2^{k-i}) + i$$

Sub $i = K$

$$A(2^K) = A(2^{K-K}) + K$$

$$= A(2^0) + K \quad [A(0) = 0]$$

$$A(2^K) = K \quad [\because K = \log n,\ n = 2^K]$$

$$\boxed{A(n) = \log n = O(\log n)}$$

x. Solve the recurrence equation of the fibonacci series $T(n) = T(n-1) + T(n-2)$, subject to $T(0)=0$, $T(1)=1$

4) Fibonacci series

Recurrence equation

$$F(n) = F(n-1) + F(n-2), \quad n > 1$$

0 1 1 2 3 5 8 13 .....

$$F(0) = 0 \quad F(1) = 1$$

$$F(n) = F(n-1) + F(n-2), \quad n > 1$$

Initial condition $F(0) = 0$, $F(1) = 1$

$$F(n) - F(n-1) - F(n-2) = 0$$

Sub $F(n) = x^n$, characteristic equation is

$$x^n - x^{n-1} - x^{n-2} = 0$$

divide by x^{n-2}

$$\frac{x^n}{x^{n-2}} - \frac{x^{n-1}}{x^{n-2}} - \frac{x^{n-2}}{x^{n-2}} = 0$$

$$x^2 - x - 1 = 0$$

$$ax + bx + c = 0 \quad r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$a = 1 \quad b = -1 \quad c = -1$$

$$r = \frac{1 \pm \sqrt{1 - 4.1 \times -1}}{2}$$

$$r = \frac{1 \pm \sqrt{5}}{2}$$

$$r_1 = \frac{1 + \sqrt{5}}{2} \quad r_2 = \frac{1 - \sqrt{5}}{2}$$

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

$$F(n) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^n + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^n$$

Initial condition $F(0) = 0 \quad F(1) = 1$

$n = 0 \quad F(0) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^0 + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^0 = 0$

$$\alpha + \beta = 0$$

$n = 1 \quad F(1) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^1 + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^1 = 1$

$$\alpha\left(\frac{1+\sqrt{5}}{2}\right)^1+\beta\cdot\left(\frac{1-\sqrt{5}}{2}\right)^1=1$$

$$\begin{cases}\alpha+\beta=0 & \text{—(1)}\\ \alpha\left(\frac{1+\sqrt{5}}{2}\right)+\beta\left(\frac{1-\sqrt{5}}{2}\right)=1 & \text{—(2)}\end{cases}$$

From (1) $\Rightarrow$ $\beta=-\alpha \Rightarrow$ (3)

Sub (3) in (2)

$$\alpha\left(\frac{1+\sqrt{5}}{2}\right)-\alpha\left(\frac{1-\sqrt{5}}{2}\right)=1$$

$$\cancel{\alpha/2}+\alpha\frac{\sqrt{5}}{2}-\cancel{\alpha/2}+\frac{\alpha\sqrt{5}}{2}=1$$

$$\cancel{2}\cdot\alpha\frac{\sqrt{5}}{\cancel{2}}=1$$

$$\boxed{\alpha=1/\sqrt{5}}$$

$$\boxed{\beta=-1/\sqrt{5}}$$

$$F(n)=1/\sqrt{5}\left(\frac{1+\sqrt{5}}{2}\right)^n-1/\sqrt{5}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$=\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

$$F(n) = \frac{1}{\sqrt{5}} \left(\phi^n - \hat{\phi}^n \right)$$

$$\phi^n = \frac{1+\sqrt{5}}{2} = 1.618$$

$$\hat{\phi}^n = \frac{1-\sqrt{5}}{2} = -0.61 \text{ [between } -1 \text{ and } 0]$$

F(n) grows exponentially.

$$\boxed{F(n) = 1/\sqrt{5}\, \phi^n}$$

Algorithm

F(n)

if $n \le 1$ return n

else return $F(n-1) + F(n-2)$