Basic operation: Most of the algo
The operation contributing the most to the total running time and ~~compute the number of times the basic~~

Let C_{op} be the execution time of algorithm's basic operation on a particular computer $C(n)$ be the ~~no~~ number of times the basic operation to be executed for an algorithm. Then the running time $T(n)$ of a program implementing this algorithm is given by

$$T(n) = C_{op} \times C(n)$$

Q) $C(n) = \frac{n(n+1)}{2}$ How much longer will the algorithm run. if we double its input size

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

$$\therefore \frac{C(2n)}{C(n)} = \frac{2(2n+1)}{(n+1)}$$

Orders of growth

Measuring the performance of an algorithm in relation with input size n is called orders of growth.

Date: / /

Asymtotic Notations

This notation is called asymtotic because it deals with behaviour of functions in the limit i.e. for sufficiently large values of its parameters.

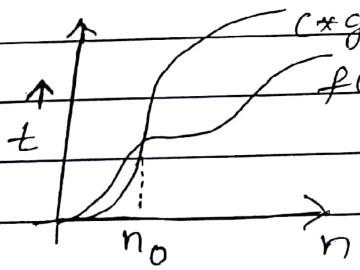
Bigoh (O)
Omege (Ω)
Theta (Θ)

Bigoh (O)

A function $f(n)$ is said to be in $O(g(n))$ if $f(n)$ is bounded above by some constant multiple of $g(n)$ for all large n i.e.

$\forall\ c, c>0$ and $n_0, n_0 \in W$ such that

$$f(n) \leq c * g(n) \quad \forall\ n \geq n_0$$



Q) $f(n) = 100n+5$, $g(n) = n^2$.
Find if $100n+5 \in O(n^2)$.

⇒ $100n+5 \leq c \cdot n^2$. for $c = 105$.
$-c \cdot n^2 + 100n + 5 \leq 0$. We can obtain $n_0 =$
$\therefore 100n+5 \in O(n^2)$

Date: / /

8) For input sizes of n. say that insertion sort run in $8n^2$ steps exactly. and merge sort runs in $64n\log_2 n$ steps exactly for what value of n does merge sort perform better than insertion sort.

-> $8n^2 \geq 64n\log_2 n$.

$n^2 \geq 8 \cdot \log_2 n$.

$n - 8\log_2 n \geq 0$.

$\frac{n}{8} \geq \log_2 n$

$2^{n/8} \geq n \quad \therefore n_0 = 44$

9)

10) $3n^3 + 2n^2 = O(n^3)$

20) $3^n \neq O(2^n)$.

10) $3n^3 + 2n^2 \leq C \cdot n^3$.

$C = 5$.

$2n^2 \leq 2n^3$

$n^2 \leq n^3$.

$\forall\ n \geq 1 \quad \therefore n_0 = 1$

$\therefore 3n^3 + 2n^2 \in O(n^3)$

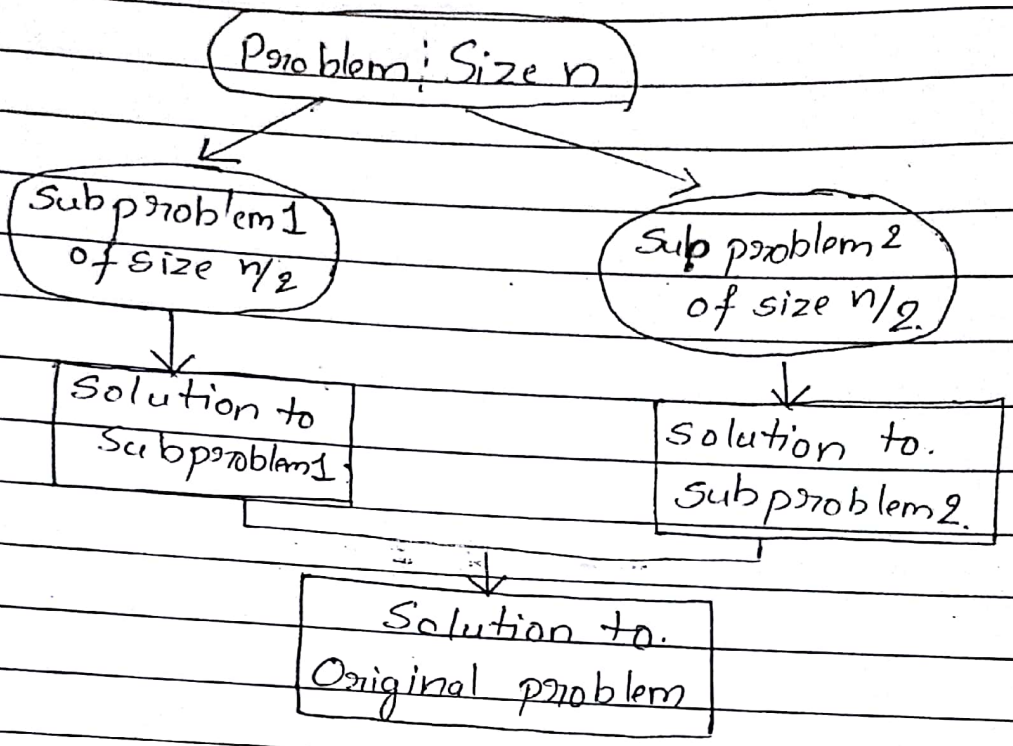
20) $3^n \leq C \cdot 2^n$

But $3^n > C \cdot 2^n$ for some $n > n_0$.

papergri

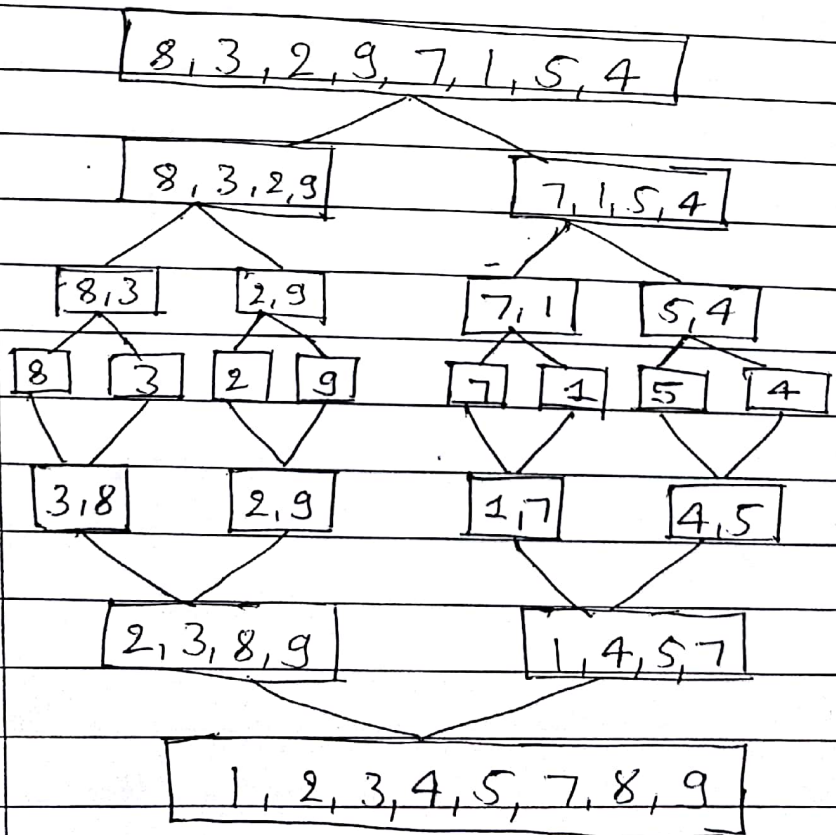
Date: / /

Divide and Conquer



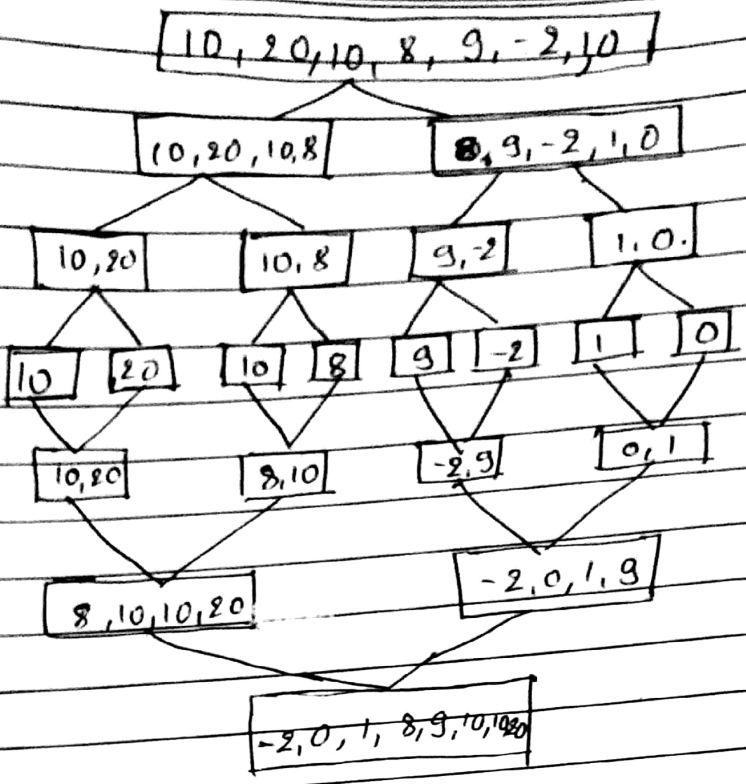
Merge Sort

1)



Date: / /

8)



Quick sort

9) low high
5, 3, 1, 9, 8, 2, 4, 7

Pivot or key = a[low] = 5.
i = low + 1
j = high.

* Increment i if pivot ≥ a[i]
* Decrement j if pivot ≤ a[j].

Once i and j can't be increment or decrement
Swap i and j and continue increment and decrement.
When j < i, Swap pivot and a[j].
Divide array into two arrays from low to j-1 and j+1 to high. Perform quicksort on these arrays.

2, 3, 1, 4, 8, 2, 4, 7

key i j

⇓

5, 3, 1, 9, 8, 2, 4, 7.

i j

⇓

5, 3, 1, 4, 8, 2, 9, 7

i j

⇓

5, 3, 1, 4, 8, 2, 9, 7

i j

5, 3, 1, 4, 2, 8, 9, 7.

i j

5, 3, 1, 4, 2, 8, 9, 7.

j i

low high low high.

2, 3, 1, 4, [5], 8, 9, 7

j

2, 3, 1, 4, [5] 8, 9, 7

key i j key i j

2, 3, 1, 4 [5] 8, 9, 7

i j i j

2, 1, 3, 4 [5] 8, 9, 9

i j i j

2, 1, 3, 4 [5] 8, 7, 9

j i j i

1, [2], 3, 4, [5], 7, [8], 9.

Q5 10, 20, 10, 8, 9, -2, 1, 0.

i j

10, 0, 10, 8, 9, -2, 1, 20

i j

10, 0, 10, 8, 9, -2, 1, 20.

i j

10, 0, 1, 8, 9, -2, 10, 20

i i

10, 0, 1, 8, 9, -2, 10, 20
i j

10, 0, 1, 8, 9, -2, 10, 20
j i

-2, 0, 1, 8, 9, [10], 10, 20.
pivot i j

Merge sort

Merge(B[0..p-1], C[0...q-1], A[0...p+q-1])
/* Merge two sorted arrays into one
*Input array B[0...p-1] & C[0...q-1] both sorted *Output sorted A[0....p+q-1] */
i ← 0, j ← 0, k ← 0
while i < p && j < q
{ if B[i] ≤ C[j].
A[k] ← B[i]; i ← i+1;
else
A[k] ← C[j]; j ← j+1;
k ← k+1 }
if i = p.
Copy C[j...q-1] to A[k...p+q-1]
else.
Copy B[i....p-1] to A[k...p+q-1]

Mergesort(A[0...n-1]).
if n > 1.
Copy A[0...([n/2]-1)] to B[0...[n/2]-1].
Copy A[n/2...n-1] to C[0...[n/2]-1]
Mergesort(B[0...[n/2]-1])
Mergesort(C[0...[n/2]-1])
Merge(B, C, A)

Quick sort

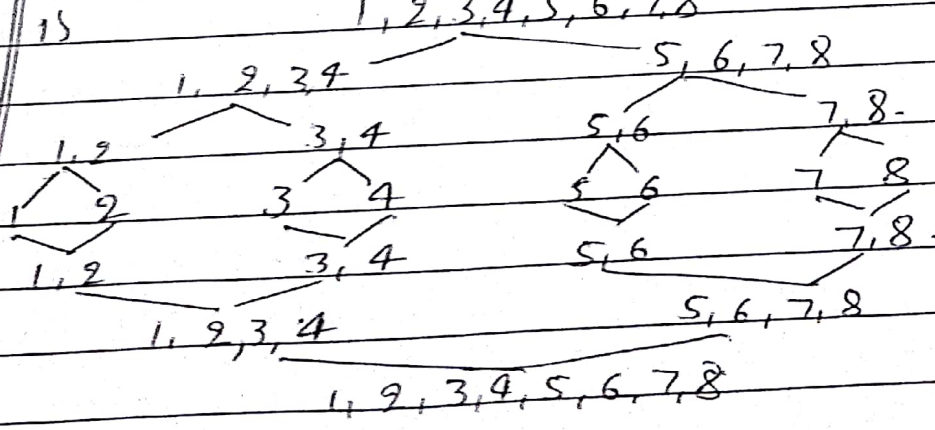
```
Quicksort ( A[l ... r] )
  if l < r.
  s ← partition (A[l ... r])
    Quicksort(A[l ..... s-1]);
    Quicksort (A[s+1 .... r]);

partition (A[l ... r]),
   p ← A[l].
   i ← l ; j ← r+1;
   repeat
    repeat i ← i+1 until A[i] ≥ p.
    repeat j ← j-1 until A[j] ≤ p.
    swap (A[i], A[j])
    until i ≥ j
   swap (A[i], p[j]); // Undo last swap when i ≥ j
   swap (A[l], A[j]);
   return j
```

Q) Perform Quick and Merge sort on.

1) 1, 2, 3, 4, 5, 6, 7, 8
2) 8, 7, 6, 5, 4, 3, 2, 1
3) -1, 2, 8, 10, 3, 0, -1, 7.

1)



1, 2, 3, 4, 5, 6, 7, 8

key i j

b) 1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

key i j

1 2 3 4 5 6 7 8

c) 8 7 6 5 4 3 2 1

key i i i i i i j

i

1 7 6 5 4 3 2 | 8

key i j

1 | 7 6 5 4 3 2 | 8

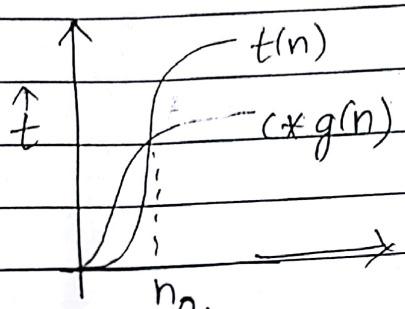
7 | 1 6 5 4 3 2 | 8

7 6 5 ✓

Omega

A function $t(n)$ is said to be in Ω of $g(n)$ ($t(n) \in \Omega g(n)$) if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large N. i.e. If there exists some positive constant c and Non negati integer N_0 such that

$t(n) \geq c * g(n) \quad \forall \; n \geq n_0$



Theta

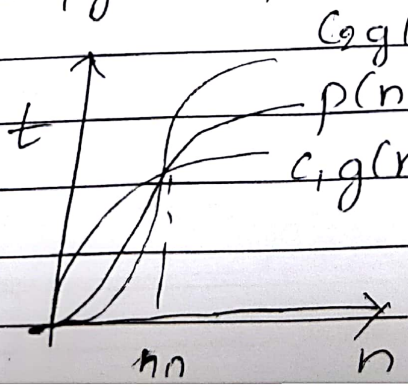
A function $p(n)$ is said to be in θ of $g(n)$ ($p(n) \in \theta g(n)$) is bounded both above and below by some positive constant multiple of $g(n)$.

$c_1 g(n) \leq p(n) \leq c_2 g(n)$ for all large

i.e.

If there exists some positive constant c_1 & c_2 and some non-Negat integer n_0, such that

$c_1 g(n) \leq p(n) \leq c_2 g(n) \quad \forall \; n \geq n$



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

Using limits for comparing orders of growth

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 \\ c>0 \\ \infty \end{cases}$$

$0 \Rightarrow t(n)$ has small order of growth than $g(n)$.
$c \Rightarrow t(n)$ has same order of growth as $g(n)$.
$\infty \Rightarrow t(n)$ has larger order of growth than $g(n)$

First two cases mean that $t(n) \in O\,g(n)$.
Las two mean that $t(n) \in \Omega\, g(n)$
Second case mean that $t(n) \in \Theta g(n)$.

L Hôspital Rule

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \lim_{n\to\infty} \frac{t'(n)}{g'(n)}$$

2) Stirling formulae.

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Basic efficiency classes

Efficiency class	order of Growth	Description	Example
Constant	1	Short of best case efficiency.	Scanning one element
Logarithmic	log n.	Algorithm doesn't consider all its input Rather problem is divided into smaller parts on each iteration	Performing Binary search operation
Linear	n	Running time of algorithms depends on i/p size. n.	Linear search
nlogn	nlogn	Some instance of i/p is considered for the list of size n	Divide & Conquer algorithms in average case
Quadratic	n^2	When algo has two nested loops this occurs	Scanning matrix elements
Cubic	n^3	When algo has three nested for loops	Matrix multiplication
Exponential	2^n	When algo has very fast rate of growth	Generating all subsets of n
Factorial	n!	Computing all Permutation	Generating Permutations

Q) Order of the following functions according to their orders of growth from lowest highest & highest to lowest
$n^2, n!, n\log_2 n, n, \log_2 n, 2^n, n^3$.

Q)
$\log_2 n, n, n\log_2 n, n^2, n^3, 2^n, n!$

$n!, 2^n, n^3, n^2, n\log_2 n, n, \log_2 n$

Q) For each of the following algorithms indicate its basic operation.
i) Computing sum of n-numbers. => Addition
ii) Computing $n!$ => Multiplication
iii) Sorting. =>
iv) Searching => } Comparison
v) Finding the largest element in a list of n numbers. => Comparison
vi) Euclid's algorithm. => Mod
vii) Sieve of Eratosthenes. => Mod.
viii) Towers of Hanoi => Movement of disk
ix) Matrix mul ↓ Addition & Multiplication
x) Pen & Pencil algo
xi) for multiplying 2 n-digit

Q) Find GCD 3,1415, 17273. = 1

Q) For each of the following functions what will happen to functions value if its argument is increased 4 fold
i) $\log_2 n$ ii) $\sqrt{n}$ iii) $\sqrt[3]{n}$ iv) n^2
-> $\log_2 4n$. $2\sqrt{n}$ $4(n)$ $16(n^2)$
$2 + \log_2 n$.

1) $\sum_{i=1}^{n} 1 = 1+1+\cdots+1 = n \in \Theta(n)$

2) $\sum_{i=1}^{n} i = 1+2+\cdots+n = \frac{n(n+1)}{2}$

3) $\sum_{i=1}^{n} i^k = 1^k+2^k+\cdots+n^k = \frac{n^{k+1}}{k+1}$

4) $\sum_{i=1}^{n} a^i = a^1+a^2+\cdots+a^n = \frac{a^{n+1}-1}{n+1}$

5) $\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$

6) $\sum_{i=1}^{n} c a_i = c \sum_{i=1}^{n} a_i$

7) $\sum_{i=l}^{u} 1 = u-l+1$

8) $f(n) = 100n+5$ express $f(n)$ using ~~Big Oh~~ omega and Theta.

$\therefore$ i) $f(n) \geq c * g(n) \quad \forall\ n \geq n_0$

let $g(n) = n$, $c = 100$.

$\therefore 100n+5 \geq 100n \quad \forall\ n \geq 1$.

$\therefore n_0$.

$\therefore 100n+5 \in \Omega(n)$.

ii) Let $g(n) = n$

Let $c_1 = 100$, $c_2 = 105$.

$\therefore 100n \leq 100n+5 \leq 105n$.

$\therefore \quad \forall\ n \geq 1 \quad \therefore n_0 = 1$.

$\therefore f(n) \in \Theta(n)$.

Q) $f(n) = 10n^3 + 5$ represent O, Ω & Θ

→ i) Let $g(n) = n^3$, $c = 20$.

$\therefore f(n) \leq c \times g(n)$

$10n^3 + 5 \leq 20n^3 \Rightarrow n_0 = 1$

$\therefore f(n) \in O(n^3)$

ii) Let $g(n) = n^3$, $c = 10$.

$f(n) \geq c \times g(n)$

$10n^3 + 5 \geq 10n^3 \Rightarrow n_0 = 1$

$\therefore f(n) \in \Omega(n^3)$

iii) $c_1 = 10$, $c_2 = 20$.

$\therefore 10 \cdot n^3 \leq 10n^3 + 5 \leq 20 \cdot n^3 \Rightarrow n_0 = 1$

$\therefore f(n) \in \Theta(n)$

Q) $f(n) = 6 \cdot 2^n + n^2$.

→ $g(n) = 2^n$.

is

Q) $n(n+1)$ and $2000n^2$

→ $f(n) = n(n+1)$ $g(n) = 2000n^2$.

$n^2 + n \quad 2000n^2$

$n \leq 1999n^2$

$1 \leq 1999n$

∴ Same order of growth.

Q) $100n^2$ and $0.01n^3$.

$g(n)$

n^2 has lower order of growth.

Q) $(\log_2 n)^2$ and $\log_2 n^2 = 2 \cdot \log_2 n$.

$\therefore (\log_2 n)^2$ has higher order of growth

Q) $(n-1)!$ and $n!$

→ $n!$ has higher order of growth

Compare the orders of growth.

Q) $t(n) = \frac{1}{2} n(n-1)$ $g(n) = n^2$

→ $\lim_{n \to \infty} \frac{\frac{1}{2}(n^2 - n)}{n^2} = \frac{1}{2} \lim_{n \to \infty} \left(1 - \frac{1}{n}\right)$

$= \frac{1}{2} \times 1$

$= 1/2$.

$\therefore$ order of growth is same.

$\therefore \lim_{n \to \infty} \frac{n^2}{\frac{1}{2}(n^2 - n)} = 2 \lim_{n \to \infty} \frac{n^2}{n^2(1 - 1/n)}$

$= 2$

Q) $\log_2 n$ & $\sqrt{n}$. Q) $n!$ & 2^n. Q) $n + 4n \log n^2$ $n^2 - n/2$

Date: / /

Mathematical Analysis of Non-Recursive Algorithm

General plan for analyzing time efficiency of non-Recursive algorithm

i) Decide on parameter indicating an i/p size
ii) Identify the algorithm's basic operation
iii) Check whether no of times basic operation is executed only on the size of an input If it also depends on additional property worst, average and if necessary best case efficiencies have to be investigated seperately.
iv) Set up a sum expressing the number of times the algorithm's basic operation is executed
v) Using standard formulas and rules of some manipulation either find a closed form formula for the count or at the very least establish its order of growth.

Ex: Finding maximum value of an Array.

```
maxval ← A[0]
for i ← 1 to n-1 do
    if A[i] > maxval
        maxval ← A[i]
return maxval
```

Comparison is the basic operation

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

$C(n) = n-1$

2) for $i \leftarrow 0$ to $n-2$ do
for $j \leftarrow i+1$ to $n-1$ do.
if $A[i] == A[j]$ return false
return true.

$\therefore C(n) = \sum_{i=0}^{n-2} 1 \left(\sum_{j=i+1}^{n-1} 1\right)$ $\quad n-1-(i+1)+1$ $\quad n-i+1$

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

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

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

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

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

3) for $i \leftarrow 0$ to n do
for $j \leftarrow 0$ to m do
sum = 0.
for $k \leftarrow 0$ to p do.
$A[i][j] = sum + B[i][k] * C[k][j];$

$\therefore \sum_{i=0}^{n} 1 \sum_{j=0}^{m} 1 \sum_{k=1}^{p} 1.$

$\rightarrow \sum_{i=1}^{n} \sum_{j=1}^{m} p. = nmp.$

Date: / /

Recurrence relation.

1) Factorial

if $n=0$ return 1

else return $F(n-1)*n$

$\therefore F(0) = 1$

$F(n) = F(n-1) + 1$

$M(n) = M(n-1) + 1$

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

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

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

$= [M(n-i)+1] + i-1$

$i = n$

$= M(n-n) + 1 + n - 1$

$= M(0) + (n)$

$M(n) = M(1)$

2) Tower of Honoi

$M(0) = 0$

$M(1) = 1$

$M(n) = M(n-1) + 1 + M(n-1)$

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

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

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

$= 2^i M(n-i) + (i-1)(i-2) + 1 \qquad i = n-1.$

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

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

Date: / /

Divide and Conquer

Master Theorem:

If $f(n) \in \Theta(n^d)$ with $d \geq 0$.
In recurrence equation

$$\boxed{T(n) = aT(n/b) + f(n)}$$

then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$