DAA

Q1 i = 1
loop (i <= 1000)
1. application code → 1000 ⇒ O(1)
2. i = i+1

Q2. i = 1
loop (i < n)
1. application code → $\log_2 n$
2. i = i*2

Q3. i = n
loop (i >= 1)
1. application code → $\log_2 n$
2. i = i/2.

ASYMPTOTIC NOTATION :-

↳ Get rid of unneeded information. (rounding off)

1,000,001 ≈ 1,000,000
$3n^2 \approx n^2$.

Rank the following functions :- → in increasing order of complexity

1) $7n^3 - 3n = O(n^3)$
2) $4n^2 = O(n^2)$
3) $n = O(n)$
4) $\log_2(n^6) = O(\log n)$ ↳ $O(1)$

5) $\frac{1}{n^4} + 18n^5 = \frac{1+18n^9}{n^4} = O(n^5)$

6) $n^8 \cdot 621909 = O(n^8)$

7) $3^n = ~~O(3^n)~~ \quad 2^{O(n)}$

8) $\log(\log n) = O(\log(\log n))$

9) $2^{3n} = ~~O(2^{3n})~~ = O(2^n) \quad 2^{O(\log^2 n)}$

10) $6n \log n = O(n \log n)$

11) $n! = O(n!)$.

~~6), 8), 4), 3), 10), 2), 1), 5), 9), 7), 11).~~

Q Algorithm Fibonacci series :-

Ans
1. INT a = 0, b = 1, c
2. loop (i = 2 ; i <= n ; i++)
{ c = a + b
a = b
b = c
}

⇒ $O(n)$.

$3^n = 2^{3n \log_2 3}$

$2^{3n} = 2^{3n \log_2 2}$

$= 2^{n \log_2 2^3} = 2^{n(\log_2 8)}$

$= 2^{\theta(n)}$

Q Write an algo whose complexity is nlogn.

Ans
```
int sum = 0;
for (int i = 0; i<n; i++)          // n+1
  for (int j = 0; j<n; j *= 2)     // (n+1).logn
    sum += j;                      // n logn.
```

Total $= n+1 + n\log n + 1 + n\log n$

$= 2n\log n + n + 2$

$= O(n\log n)$

Insertion sort :-

8	2	4	9	3	6	# less than (compare) than swap.
2	8	4	9	3	6	
2	4	8	9	3	6	
2	4	8	9	3	6	
2	3	4	8	9	6	
2	3	4	6	8	9	

```
for (int i = 1; i < n; i++)
{ int j = i-1;
  int x = arr[i];
```

```
while (arr[j] > x && j >= 0)
{ arr[j+1] = arr[j];
    j--;
}
arr[j+1] = x;
}
```

OR

```
insertion_Sort (A)
{
    for (j = 2 to A.length)          (N+1-2)
        key = A[j]                   (1)
    //Insert A[j] into sorted sequence A[1..j-1]
    i = j-1                          (1)
    while ( i>0 and A[i] > key)      (N)(N-1)
        A[i+1] = A[i]                (1)
        i = i-1                      (1)
    A[i+1] = key.                    (1)
}
```

$$F(n) = (N+1-2)+1+1+N(N-1)+1+1+1$$
$$= N^2 - N + (N-1) + 5$$
$$= N^2 + 4$$

9, 7, 6, 5, 4, 3 Swapping → comparison.

2 → 1 + 1 = 2(1)
3 → 2 + 2 = 2(2)
4 → 3 + 3 = 2(3)
5 → 4 + 4 = 2(4)
6 → 5 + 5 = 2(5)

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

$= n^2 - n$

$F(n) = O(n^2)$ → Reverse sorted.

Big O : WORST CASE ÷ It is the longest running time for input of size n.

O represents upper bound / asymptotically upper bound.

Eg : 1 2 3 4 5 6
→ comparison.

2 ÷ 1 = 1(1)
3 : 1 = 1(1)
4 : 1 = 1(1)
5 : 1 = 1(1)
6 : 1 = 1(1) → (n-1) times.
1 + 1 + 1 + ~~1~~ = (n-1)

$F(n) = \Omega(n)$ (Best case)

~~$\Omega(f(n))$~~ BEST CASE :- It is the time taken for some input data set that results in best possible manner.
You cannot do better. Asymptotically Lower Bound.

$f(n) = \Omega(g(n))$

Not taught ~~$f(n) \geq c \cdot g(n), \; n > n_0$~~.

$\Theta(f(n))$ AVERAGE CASE :- → Average performance
→ Asymptotically tight bound.

Binary Search :- $O(\log n)$.

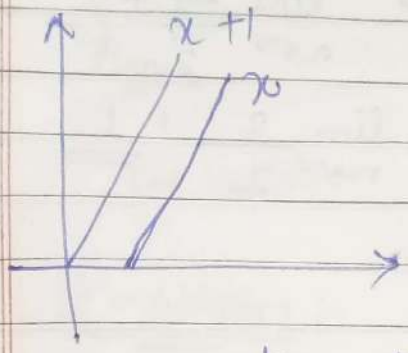
1 2 3 4 5 6

4 ≡ 1 : 1 $\Omega(1)$

1 ≡ 1 : 1
2 : 1
1 + 1 = 2 : $\approx \lfloor \log 6 \rfloor$

For N elements
1 + 1 + ... $\log N$ times $\approx \log N$
$O(\log N)$.

ASYMPTOTIC : A line that continuously approaches a given curve but does not meet it at any finite distance.

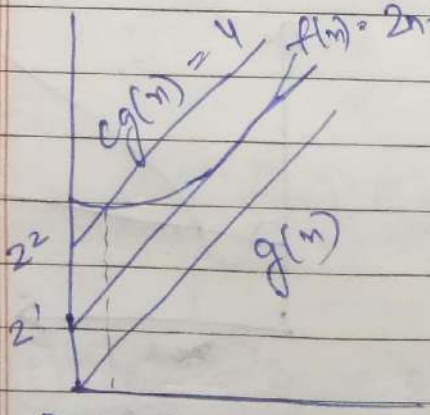


x is asymptotic with x+1.
We describe the behaviours of function in terms with limit.
Let f(x) & g(x) be functions, from the set of real nos, we say that f and g are asymptotic i.e. f(x) ≈ g(x) if

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = \text{constant}$$

Big-Oh : Given functions ~~f(x)~~ f(n) and g(n), we say that f(n) is O(g(n)) if and only if there are ~~two~~ +ve constants c and ~~x~~ n_0, s.t.

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



$\Rightarrow 2n + 6$ is $O(n)$.

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some constant $0 \leq c \leq \infty$

g(n) is asymptotic upper bound

X

1) $n^3 = o(n^2)$ → wont satisfy condition as $c \to \infty$.

✓ 2) $n^2 + n = O(n^2)$

$$\lim_{n \to \infty} \frac{n^2+n}{n^2} = \lim_{n \to \infty} \frac{2n+1}{2n}$$

$$= \lim_{n \to \infty} \frac{2}{2} = 1.$$

$c = 1$

and now $n_0 = 0$

$n^2 + n \leq cn^2$

$n^2(1-c) + n \leq 0$

$n^2 + n \leq 1 \cdot n^2$

$\therefore n \leq 0$

Ω - NOTATION :-

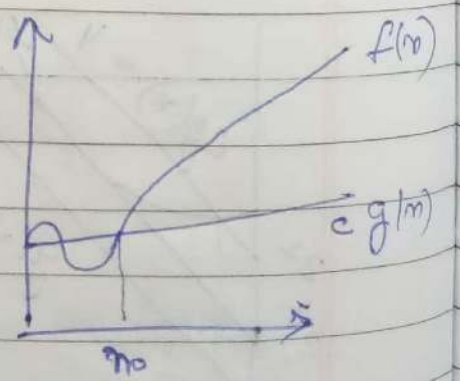
$$c\,g(n) \leq f(n) \quad \forall n \geq n_0$$

$g(n)$ is asymptotic lower bound.

for some $0 < c \leq \infty$

1) $n^3 = \Omega(n^2)$
2) $n^0 = \Omega(n^2)$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$



$f(n) = \Omega(g(n))$

1) $\lim_{n \to \infty} \frac{n^3}{n^2} = \infty$.

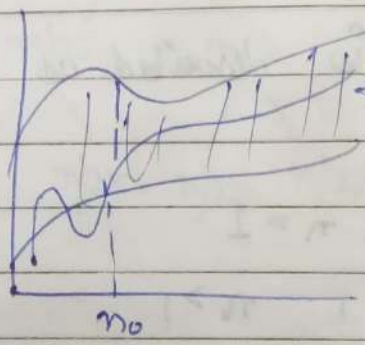
Valid

2) $\lim_{n \to \infty} \frac{n}{n^2} = 0$ ⇒ does not satisfy cond^n.

Not valid

Θ – Notation :-

$\Theta(g(n)) = \{ f(n)$, +ve constants c_1, c_2 and n_0 s.t. $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\forall\ n \geq n_0 \}$



$0 \leq c_1 \leq \infty$

$0 \leq c_2 < \infty$.

Set of all functions that have the same rate of growth as $g(n)$, $g(n)$ is asymptotic tight bound for $f(n)$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for $\boxed{0 < c < \infty}$

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

$$\lim_{n\to\infty} \frac{\frac{n^2}{2} - 2n}{n^2} = \lim_{n\to\infty} \frac{n - 2}{2n}$$

$$\lim_{n\to\infty} \frac{1}{2} = \frac{1}{2} = c.$$

DIVIDE AND CONQUER :-

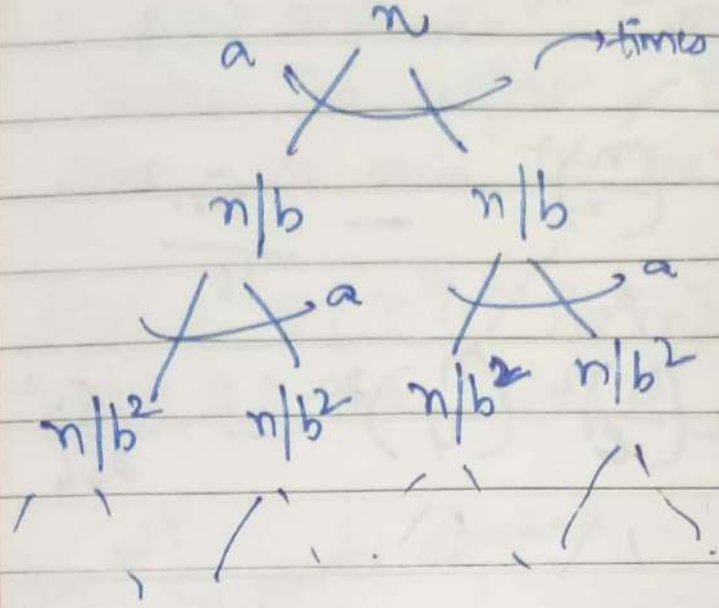
Here we split the inputs into k distinct subsets, $1 < k \leq n$ yielding k subproblems. These subproblems must be solved, and then a method must be found to combine subsolutions into a solution.

Computing time is described as recurrence relation,

$$T(n) = \begin{cases} T(1), & n = 1 \\ aT(n/b) + f(n), & n > 1 \end{cases}$$

$f(n)$ is the time for dividing the problem and combining the solutions of subproblems.

$T(n) = aT(n/b) + (n)$ → time to combine sub soln into soln of that particular level.



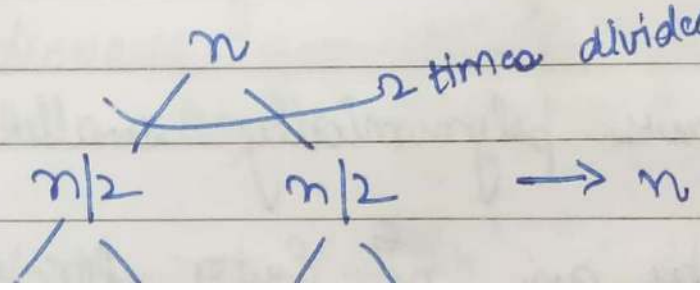
→ times (a times divided)

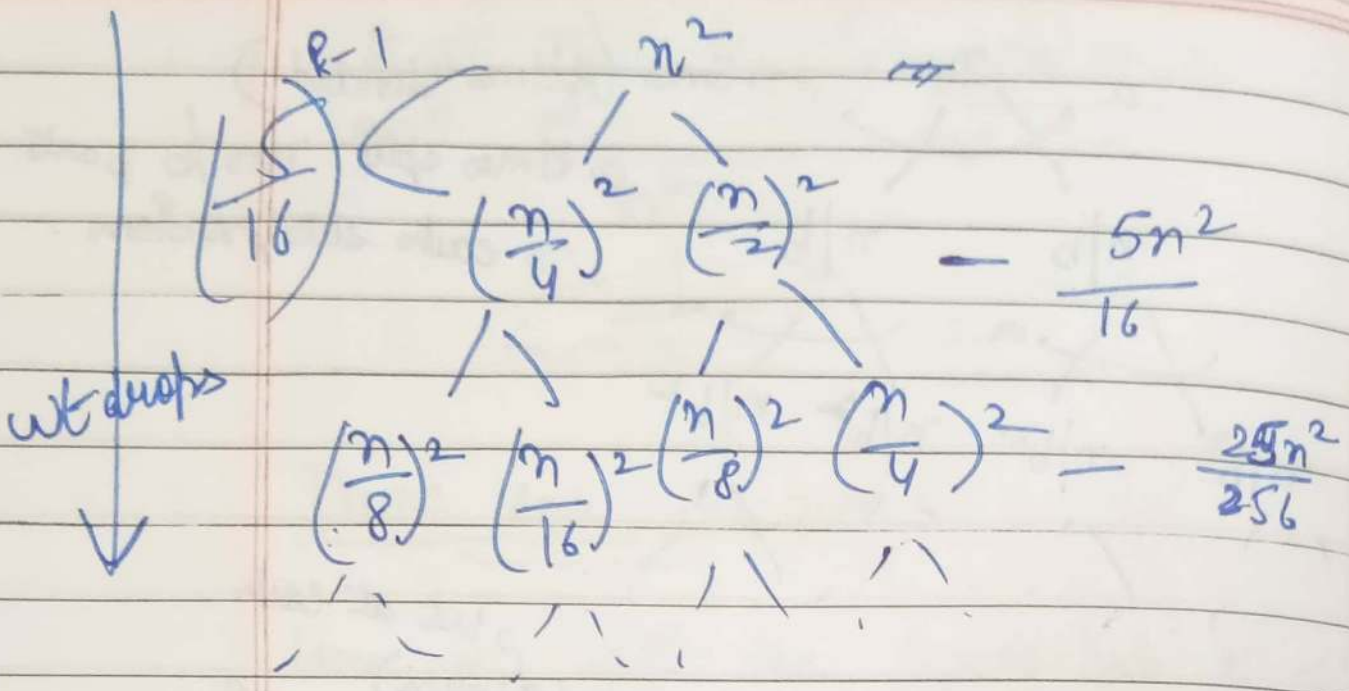
a times split into n/b parts each sub problem.

→ wt at each

→ O(n log n) levels.

$T(n) = 2T(n/2) + n$ (Merge Sort)





MASTER'S THEOREM :-

Case 1: $f(n)$ grows polynomially smaller than $n^{\log_b a}$ by an n^{ϵ} factor for some constant $\epsilon > 0$

wt ↓

inc

→ equal wt → all considered → n × height tree

Case 2: $f(n)$ and $n^{\log_b a}$ grows at similar rates. The weight is approximately the same on each at the $\log_b n$ levels.

$f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$, Then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

→ dec wt → ~~root~~ dominate

Case 3: $f(n)$ grows polynomially faster than $n^{\log_b a}$ by an n^{ϵ} factor. The weight decreases geometrically from the root to the leaves. ~~f(n) = Ω(log)~~

$f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and $f(n)$ satisfies the regularity condition that $af(n/b) \leq cf(n)$ some constant $c < 1$. $T(n) = \Theta(f(n))$ for

Q1. $T(n) = 8\,T(n/2) + n^2$
$T(n) = 2T(n/2) + \log n$
$T(n) = 9T(n/3) + n$

→ all are case 1

A1 ① $a = 8$
$b = 2$

$\Theta(n^{\log_2 8}) = \Theta(n^3)$

$n^{(\log_2 8)-\epsilon} = n^2$

$\log_2(8) - \epsilon = 2$ $\qquad n^{\log_b a - \epsilon}$

$8 - \epsilon \qquad \epsilon = 4$

$O(n^3)$.

(2) $a = 2 \qquad b = 2$

$n^{\log_2(2)-\epsilon} = (\log n)$

↳ not possible.

$n^{1-\epsilon} = \log n$

ϵ not possible.

$1 - \epsilon = \log_n \log n$

$\epsilon = 1 - \log_n \log n$

(3) $a = 9, \quad b = 3$

$n^{\log_3 9 - \epsilon} = n$

$2 - \epsilon = 1$

$\epsilon = 1$.

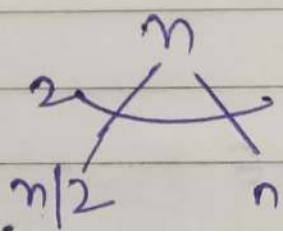
$\Rightarrow$ Complexity $= n^2$.

Computing Time of Divide and conquer is given by the Recurrence Relation :-

$$T(n) = \begin{cases} T(1), & n = 1 \\ aT(n/b) + f(n), & n > 1 \end{cases}$$

$f(n)$ is the time for dividing and computing the solution of subproblems

$$T(n) = 2T(n/2) + n$$



$\Rightarrow n\left(\frac{1}{2} + \frac{1}{2}\right) = n.$

$f(n) = \log n$

$a = 2, \; b = 2$

$n^{\log_2 2 - \epsilon} = n^{1-\epsilon}$

$n^{1-\epsilon} = \log n$

$n^{1-\epsilon} = (n-1) - \frac{1}{2}(n-1)^2 + \frac{1}{3}(n-1)^3 + \frac{1}{4}(n-1)^4 + \cdots$

$n^{-\epsilon} = 1 - \frac{1}{n} - \frac{1}{2}\frac{(n-1)^2}{n} + \frac{1}{3}\frac{(n-1)^3}{n} + - - -$

$\epsilon > 0$

$\exists$ some ϵ s.t. for various values of ϵ s.t. $\epsilon > 0$ and $n^{1-\epsilon} = \log n$

$\therefore T(n) = \Theta(n)$

$T(n) = 9\,T(n/3) + n$

$= n$

$9 \times \frac{n}{3} = 3n$ ↓ 9 ways

$\frac{n}{3} \quad \frac{n}{3}$

$a = 9, \quad b = 3.$

$n^{\log_3 9} = n^{2-\epsilon}$

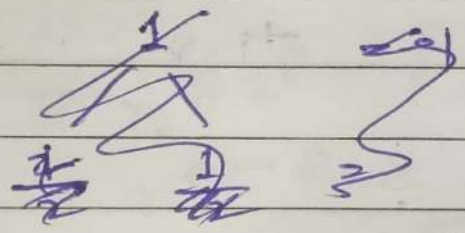
$2 - \epsilon = 2$

$\epsilon = 0.$

$T(n) = \theta(n^2)$

(4) $T(n) = \theta(n \log n)$ $k = 0.$ Case 2

(5) $T(n) = T(n/2) + 1$



$f(n) = c$

$k > 0$

$a = 9, \; b = 3$

$n^{\log_3 9 + \epsilon} = n^3$

$n^{2+\epsilon} = n^3$

$2 + \epsilon = 3$

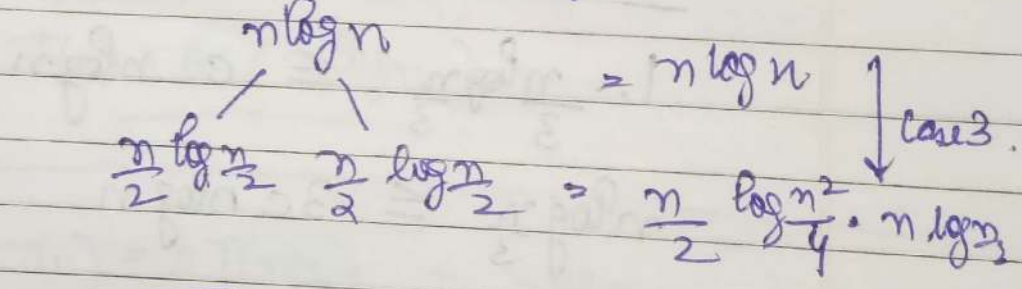
$\boxed{\epsilon = 1}$

$9x \cdot \frac{n^3}{27} \leq c\, n^3$

$\frac{n^3}{3} \leq cn^3 \Rightarrow \boxed{c = \frac{1}{3}}$

$T(n) = \Theta(n^3)$

(6) $T(n) = 2T(n/2) + n\log n$



$n^{\log_2 2 + \epsilon} = n \log n$

$n^{1+\epsilon} = n \log n$

Some value exists

$T(n) = \Theta(n \log n)$

$\boxed{k = 1}$

case 2

$n \quad n\log n$

Some c exist.

$\therefore \frac{n}{3}\log\frac{n}{3} \leq c \cdot n\log n$

$n\log\frac{n}{3} \leq 3cn\log n$

$n(\log n - \log 3) \leq 3cn\log n$

$T(n) = 3T(n/4) + n\log n$

Case 3

$a = 3, \quad b = 4$

3. $\frac{n}{4}\log\left(\frac{n}{4}\right) \leq c \cdot n\log n$

$\frac{3n}{4}\log n - \frac{3n}{2}\log 4 \leq c\, n\log n$

$n\log n\left(\frac{3}{4} - c\right) \leq \frac{3n}{2}$

$\boxed{c = \frac{3}{4}}$

$T(n) = \theta(n\log n)$

Solve: $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

Here a and b are not constants

So solve it by using **changing variables**

$m = \log_2 n$ or $n = 2^m$

$$T(n) = \sqrt{2^m}\, T(\sqrt{2^m}) + 2^m$$

$$T(2^m) = 2^{m/2}\, T(2^{m/2}) + 2^m$$

~~$T(2^m) \Rightarrow T(2^{m/2}) + 2^{m/2}$~~

~~$\frac{T(2^m)}{2^{m/2}}$~~

$$T(2^m) = 2^{(m - m/2)}\, T(2^{m - m/2}) + 2^m$$

$$\frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 1$$

$\frac{T(2^m)}{2^m}$ be $S(m)$

$$S(m) = S(m/2) + 1. \quad \text{Case 2}$$

$a = 1$

$b = 2.$

$S(m) = \theta(\log m)$

$\frac{T(2^m)}{2^m} = \theta(\log \log n)$

$T(n) = n \log(\log n)$ ← height of tree

$\frac{T(n)}{n} = \log(\log n)$

ITERATIVE SUBSTITUTION :-

$T(n) = 2T(n/2) + n$

$= 2\left[2T(n/2\times2) + \frac{n}{2}\right] + n = 4T(n/4) + 2n$

$= 4\left[2T(n/4\times2) + \frac{n}{4}\right] + 2n = 8T(n/8) + 3n$

$T(n) = 2^i T(n/2^i) + in$ (generalized eqn)

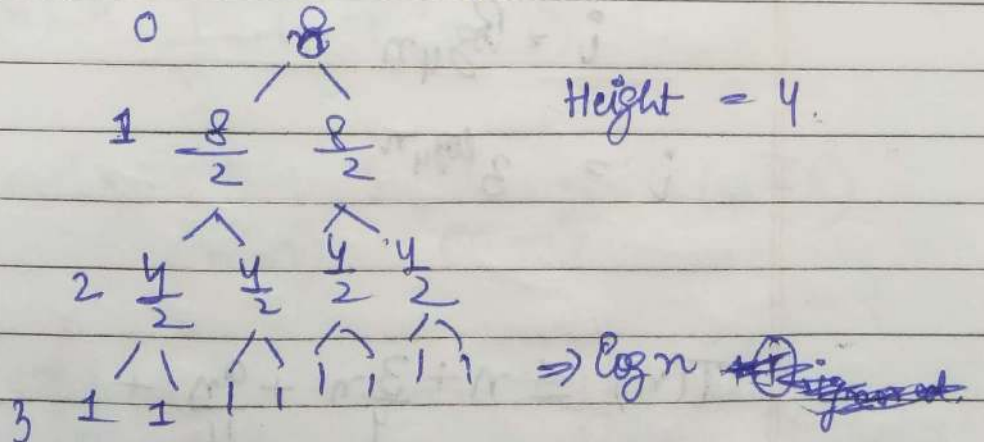
The series terminates when $\frac{n}{2^i} = 1 \Rightarrow n = 2^i$

or $i = \log_2 n$ ~ height of a tree.

$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + n\log n \quad (i = \log n)$

$= nT(1) + n\log n \Rightarrow n\log n$

for $n = 8$ draw a tree.



Height = 4.

$\Rightarrow \log n$

Q $T(n) = 3T(n/4) + n$

Ans $T(n) = 3T(n/4) + n$

$= 3\left[3T(n/4\times4) + \frac{n}{4}\right] + n$

$= 9T\left(\frac{n}{16}\right) + \frac{7n}{4}$

$= 9\left[3T\left(\frac{n}{16\times4}\right) + \frac{n}{16}\right] + \frac{7n}{4}$

$= 27T\left(\frac{n}{64}\right) + \frac{9n}{16} + \frac{7n}{4}$

$= 27T\left(\frac{n}{64}\right) + \frac{37n}{16}$

$1 + \frac{7}{37}$

$T(n) = 3^i T\left(\frac{n}{4^i}\right) + \sum_{i=0}^{\log_4 n - 1} \frac{3^i}{4^i} n$

$i = \log_4 n$

$i = 3^{\log_4 n}$

$T(n) \le n + \frac{3}{4}n + \frac{9n}{16} + \cdots + 3^i T\left(\frac{n}{4^i}\right)$

The series terminate when $T\left(\frac{n}{4^i}\right) = 1$

$\Rightarrow i = \log_4 n$

$$T(n) \leq n + \frac{3n}{4} + \frac{9n}{16} + \cdots + 3^{\log_4 n} T(1)$$

$$\leq n + \frac{3n}{4} + \frac{9n}{16} + \cdots + 3^{\log_4 n} \theta(1)$$

$$\leq n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + \theta\left(n^{\log_4 3}\right)$$

$\log_4 3 \leq 1$

$\Rightarrow O(n) \rightarrow$ worst-case

$$T(n) \leq n * \frac{1}{\left(1 - \frac{3}{4}\right)} + O(n) \quad \Rightarrow n \to \infty$$

$$\leq 4n + O(n)$$

$$\Rightarrow T(n) = \underline{\underline{O(n)}}$$

Q $T(n) = \frac{1}{n} + T(n-1)$ (Iterative subs^n)

Ans $T(n) = \frac{1}{n} + T(n-1)$

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

$$= \left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2}\right) + T(1)$$

$$= \int_0^{n-1} \frac{1}{(n-x)}\, dx + T(1)$$

$$T(n) = \log n + T(1)$$

$$= \log n + \Theta(1)$$

$$T(n) = \Theta(\log n)$$

STRASSEENS

MATRIX MULTIPLICATION :-

Let A and B be 2 $n \times n$ matrix

$C = A \cdot B$

$n \times n$

$$C(i,j) = \sum_{1 \le k \le n} A(i,k) \cdot B(k,j)$$

$\forall$ i & j between 1 to n

$C(i,j)$ → requires n multiplications

& C has n^2 elements ⇒ Matrix Multiplication $(\Theta(n^3))$

Divide & Conquer Strategy :-

To compute product of two $n \times n$ matrices,

we assume $n = 2^k$ (& n is a power of 2)

If not we add dummy row & column for 2

A & B are divided into 4 square sub-matrices each submatrix having

dimensions $\frac{n}{2} \times \frac{n}{2}$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$C_{11} = A_{11} B_{11} + A_{12} B_{12}$

$C_{12} = A_{11} B_{12} + A_{12} B_{22}$

$C_{21} = A_{21} B_{11} + A_{22} B_{21}$

$C_{22} = A_{21} B_{12} + A_{22} B_{22}$

if $n = 2$, it is computed directly.

for $n > 2$, the elements of C can be computed using matrix multiplication & addition applied to matrices of size $\frac{n}{2} \times \frac{n}{2}$.

→ Since n is a power of 2, it can be recursively computed using $n \times n$ case.

→ Continue applying till $n = 2$, so that it is computed directly.

→ To compute AB using C_{11}, C_{12}, C_{21}, C_{22} we need 8 multiplications & 4 additions of $\frac{n}{2} \times \frac{n}{2}$ matrices.

→ Horowitz & Saini

→ Cormen

Since two $\frac{n}{2} \times \frac{n}{2}$ can be added in time ~~time~~ cn^2 for some constant

Its D & C recurrence is given by

$$T(n) = \begin{cases} b, & n \leq 2 \\ 8T(n/2) + cn^2, & n > 2 \end{cases}$$

b, c ∈ constant

$cn^2 \longrightarrow cn^2$

$c(\frac{n}{2})^2 \quad c(\frac{n}{2})^2 \rightarrow 8 \cdot c\frac{n^2}{4} = c\frac{n^2}{2}$

$c(\frac{n}{4})^2 \; c(\frac{n}{4})^2 \; c(\frac{n}{4})^2 \; c(\frac{n}{4})^2 \rightarrow 16c\frac{n^2}{16} = cn^2$

dominating multiplication reduce that

$O(n^3)$

⇒ Strassens made 8 multiplications to 7 multiplications.

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$P_1 = a(f - h)$

$P_2 = (a+b)h$

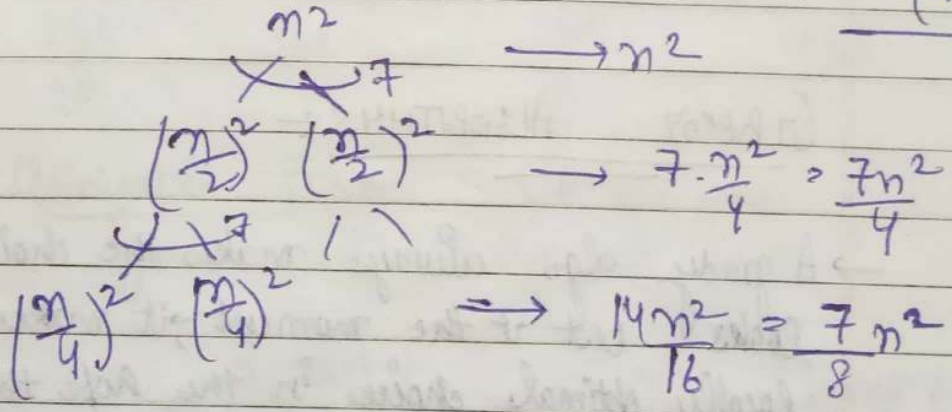
$P_3 = (d+c)e$

$P4 = d(g-e)$
$P5 = (a+d)(e+h)$
$P6 = (b-d)(g+h)$
$P7 = (a-c)(e+f)$

$r = P5 + P4 - P2 + P6$
$s = P1 + P2$
$t = P3 + P4$
$u = P5 + P1 - P3 - P7$

7 multiplications & 18 additions

$T(n) = 7T(n/2) + n^2$ $\quad T(n) = O(n^{\log 7})$



$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

Ans $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

$P1 = 1(6-2) = 4$
$P2 = 3 \cdot 8 = 24$
$P3 = 7 \cdot 5 = 35$

P4 = 8

P5 = ~~200~~ 65

P6 = -30

P7 = -22

~~65~~ 73

$r = \frac{~~208~~ - 24 ~~+~~ \div 30}{}$

$r = 19$

$s = 28$

$t = 43$

$u = 69 - 35 + 22$

$=$

65
54
9

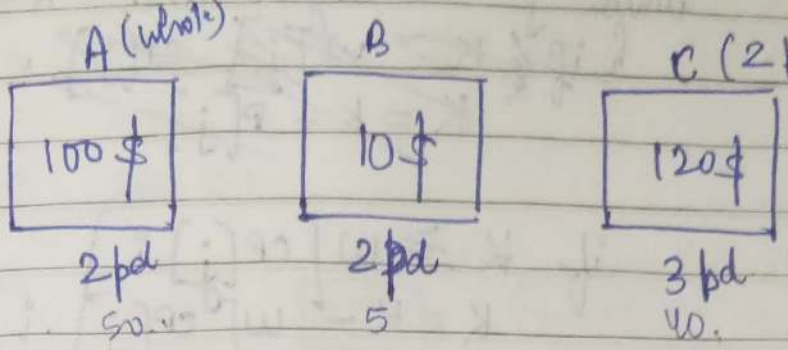
69
35
34

GREEDY ALGORITHMS :-

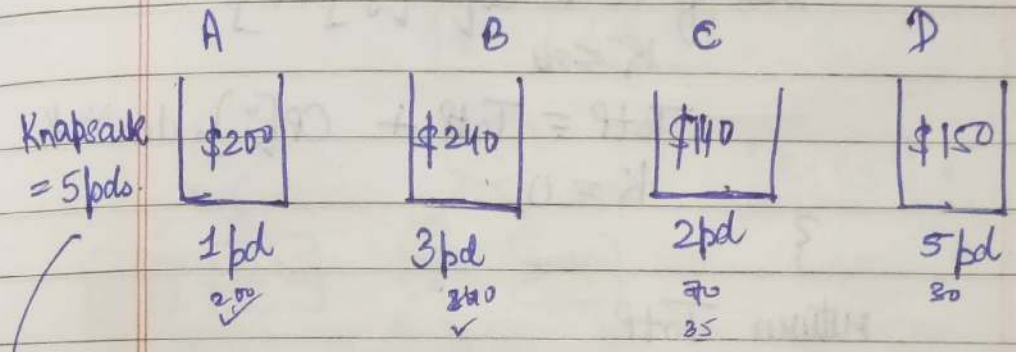
→ A greedy algo always makes the choice that looks best at the moment → it makes locally optimal choice in the hope that ~~that~~ this choice will lead to globally optimal soln.

→ Greedy algo do not always yield optimal soln but for many problems they do. Examples are Dijkstra's Shortest Path Algo and Kruskal's / Prim's Algo.

Fractional Knapsack Algo :-



if Knapsack holds 4 pds (wt) what should you steal to maximise profit.



Optimization Problem :- i.e. not just gives you a sol^n but the best sol^n.

→ O(nlogn)

Fractional Knapsack (W, P, K)

Create an array CP[ ]

struct { float den; int i; }

for j = 1 to W.length.

CP[j].den = P[i] / W[i]

CP[j].i = j

// Sort the array CP on basis of den.

Sort (CP) // in descending order.

~~while K ≠ 0~~

```
j = 1 , TotP = 0
while j < cp.length and K ≠ 0
  { if ( K > w[cp[j]] ) cp[j].
      K = K - P[j]

  if K ≥ w[cp[j].i]
      K = K - w[cp[j].i]
      TotP = TotP + P[cp[j].i]

  else if K < w[cp[j].i]
      K = 0
      TotP = TotP + cp[j].den * K
      K = 0
  }
return TotP.
}
```

→ O(n log n)

Huffman Codes :-

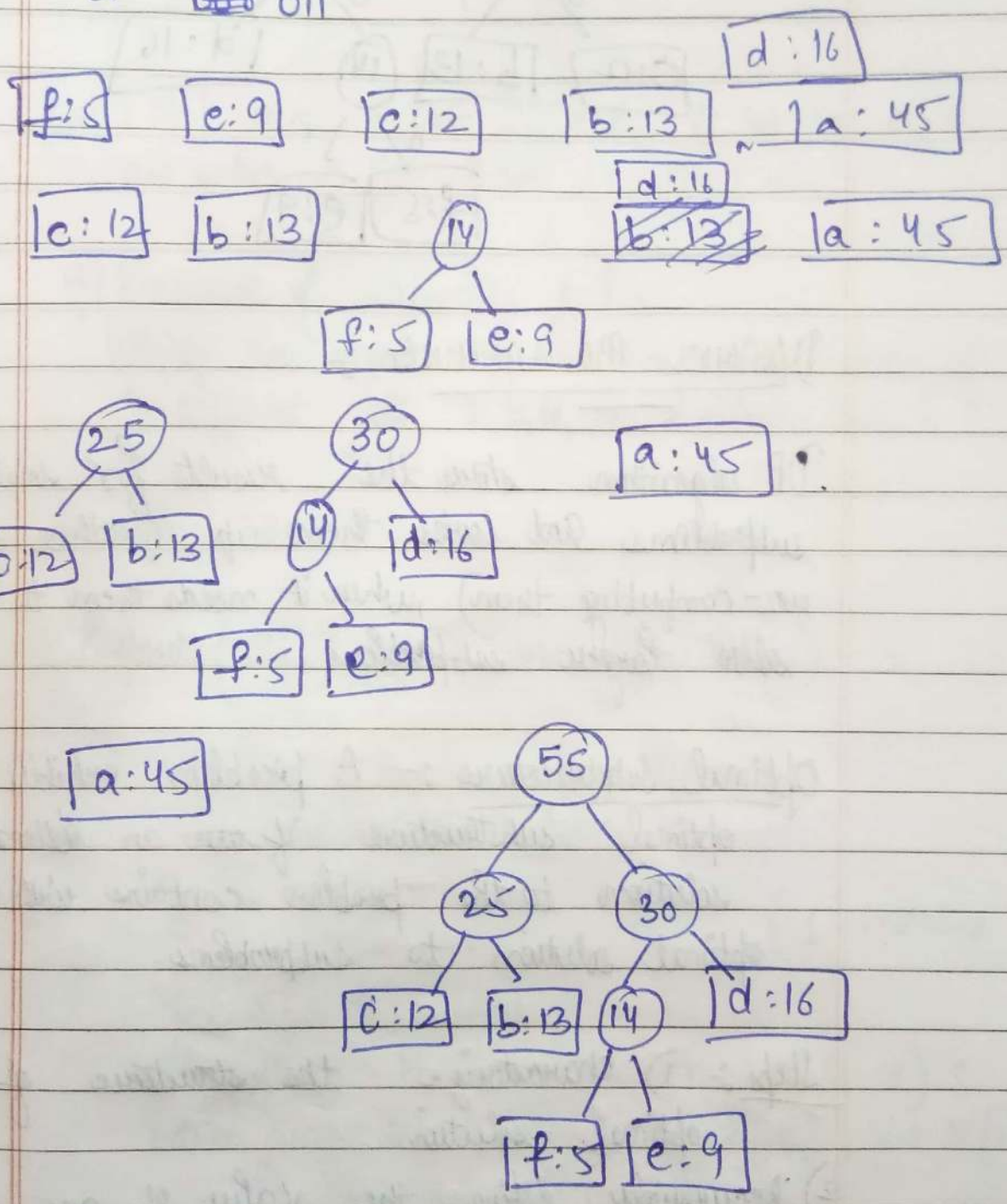
	a	b	c	d	e	f
Freq	45	13	12	16	9	5

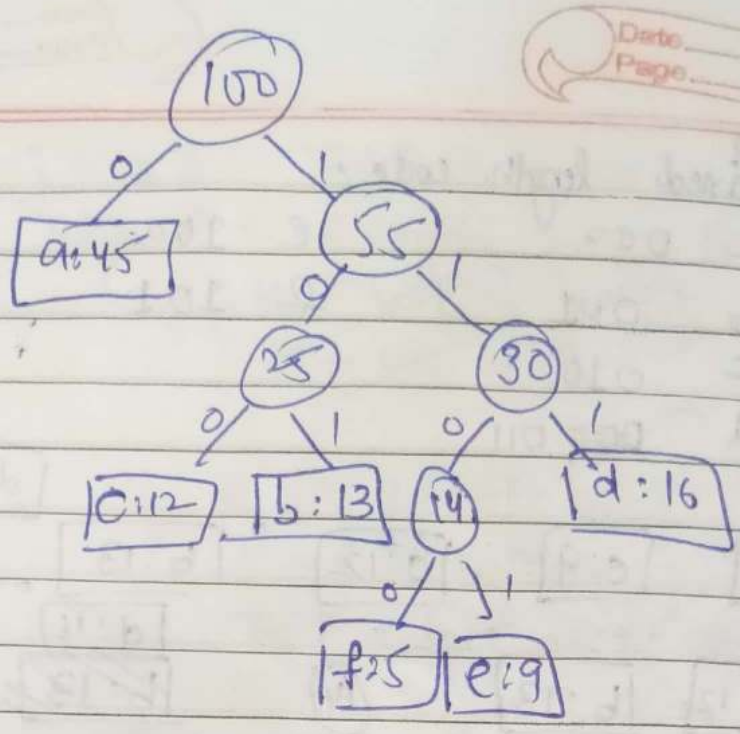
Variable length code (prefix code)

a 0
b 101
c 100
d 111
e 1101
f 1100

fixed length code:

a	000	e	100
b	001	f	101
c	010		
d	011		





DYNAMIC PROGRAMMING :-

DP algorithm stores the results for small subproblems and looks them up, (rather than re-computing them), when it needs them to solve larger subproblems.

Optimal substructure :- A problem exhibits optimal substructure if an optimal solution to the problem contains with its optimal solution to subproblems.

Steps :- 1) Characterize the structure of an optimal solution.
2) Recursively define the value of an optimal sol^n.
3) Compute the value of an optimal sol^n in a bottom-up fashion.
4) Construct an optimal sol^n from computed info.

0/1 KNAPSACK :-

: Let $U = \{u_1, u_2 - - u_n\}$ be a set of n items to be packed ~~into~~ in a knapsack of size x.

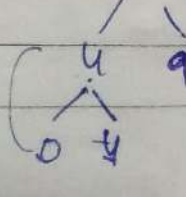
for $1 \leq j \leq n$ let S_i and V_j be the size and value of j^{th} item

a) Knapsack of capacity 9 which we want to ~~pack~~ pack with items of 4 different ~~sizes~~ sizes 2, 3, 4, 5 & values 3, 4, 5, 7.

Recursive Code:

```
Knapsack (S[], V[], n, Q, value)
{
  if (Q == 0)
      return value;
  if (n == 0)
      return -1                    (1-indexing)
  if (S[n-1] > Q)
      return Knapsack(S, V, n-1, Q, v);
  return max(Knapsack(S, V, n-1, Q - V[n], V + V[n]),
             Knapsack(S, V, n-1, Q, v));
}
```

7



Capacity

First No. of items	i,j	0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	3	3	3	3	3	3	3	3
	2	0	0	3	4	4	7	7	7	7	7
	3	0	0	3	4	5	7	8	9	9	12
	4	0	0	3	4	5	7	8	10	11	12

Let $V[i,j]$ be the value obtained by filling a Knapsack of size j with items taken from first i items $\{u_1, u_2 \dots u_i\}$ in an optimal way.

$i \geq 0, j \geq 0 \Rightarrow 0$

$i \geq 1, j \geq 1$

if ($s[1 \dots i] > \text{capacity}$) $\rightarrow 0$

else

Table[i,j] = max (Table[i-1][j] → not included, Table[i-1][j - s[i]] + v[i] → included) ; chuk if $j >= s[i]$

Table[0,j] = 0

Table[i,0] = 0

$$V[i,j] = \begin{cases} 0, & i=0 \text{ or } j=0 \\ V[i-1,j], & j < s_i \\ \max(V[i-1,j], V[i-1,j-s_i] + v_i), & i>0 \text{ \& } j \geq s_i \end{cases}$$

```
KNAPSACK (s, v, q)
{
    DP[length[s]+1, q+1]        (1-indexing in s & v)
    for i=>0 to length[s]
    { for j=>0 to q
      { if i=0 or j=0
            DP[i,j] = 0
        else if j < s[i]
            DP[i,j] = DP[i-1,j]
        else
            DP[i,j] = max(DP[i-1,j], DP[i-1][j-s[i]] + v[i]);
      }
    }
    return DP[length[s], q]        //Result.
}
```

Time Complexity = $\underline{\theta(nc)}$

```
i ← n , j ← c
Val ← V[i,j]
while ( i >= 1 and j >= 1)
{ if V[i-1, j] = Val
      i--;
  else break;
}
```

```
Declare bool mark[n] = false.
mark[i] = true
while (i >= 1 && j >= 1)
{   i--;
    j
```

LONGEST COMMON SUBSEQUENCE :-

Given a sequence $X = \langle x_1, x_2 \dots x_m \rangle$, another sequence $Z = \langle z_1, \dots z_k \rangle$ is a subsequence of X, if ∃ a strictly increasing sequence $\langle i_1, i_2 \dots i_k \rangle$ of indices of X s.t. for all $i = 1, 2, \dots k$ we have $x_i = z_j$.

Eg: $\langle A, B, C, B, D, A, B \rangle$, subsequence is $Z = \langle B, C, D, B \rangle$ with indices 2, 3, 5, 7

Def^n of LCS

→ Given two sequence X and Y, we say that a sequence Z is a common subsequence of X & Y, if Z is a subsequence of both X & Y.

The longest common subsequence (LCS) problem is the problem of finding, for given two sequences $X = \langle x_1, x_2 \dots x_m \rangle$ and $Y = \langle y_1, y_2 \dots y_n \rangle$, a maximum length common subsequence of both X & Y.

X = BDCABA
Y = ABCBDAB

↓

4

BCAB

A = zxyxyz
B = xyyzx

↓

3

xyz

Table [m+1] [n+1]

Table [0] [i] = 0 = Table [i] [0]

$$\text{Table}[i][j] = \begin{cases} \text{Table}[i-1][j-1] + 1, & X_i = Y_j \\ \max(\text{Table}[i-1][j], \text{Table}[i][j-1]), & X_i \neq Y_j \end{cases}$$

AB

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	1	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

```
Lcs (X, Y)
{
   DP[length[X]+1][length[Y]+1]
   for i=0 to length[X]
   {
      for j=0 to length[Y]
      {
         if(i==0 || j==0) DP[i][j]=0
         else if (X[i-1] == Y[j-1])
            DP[i][j] = DP[i-1][j-1]+1
         else
            DP[i][j] = max(DP[i-1][j],
                           DP[i][j-1]);
      }
   }
   return DP[length[X]+1][length[Y]+1];}
```

u, l, d

for getting the string, start from length[x] & length[y]

if max of [i-1,j] & [i,j-1] is not equal to [i,j] then take x[i], y[j] else move ahead.

```
TCS_returnsubstr(DP[][], x, y)
{
    string OP = "";
    for(i = length[x] to 0
    { for j = length[y] to 0
        { if( DP[i][j] != max(DP[i-1][j], DP[i][j-1])
            OP += x[i-1];
          else break;   // as never reach j possible.
        }
    return OP;
}
```

Store direction of movement u, l, d for moving in matrix, using that move in matrix & include only if d.

MATRIX CHAIN MULTIPLICATION :-

1) The Matrix ~~chain~~ Multiplication ~~a to~~ $a_{M_1}b \times c$ only if $b = c$ in which case the result is $a_M d$.

2) The case of multiplication $a_{M_1}b \times b_{M_2}c \times c_{M_3}d$ is abe.

The cost of ~~complete~~ compatible sequence ~~of~~ of matrix multiplications depends on its schedule (the particular multiplication ~~time~~ used).

⇒ Find order of Matrix Multiplications that minimizes the cost.

A	A1	10×100
B	A2	100×5
C	A3	5×30

$(AB)C$ ~~(A B) A~~ $A(BC)$

$\Downarrow$ $\Downarrow$

$5000 + 30 \times 5$ $100 \times 5 \times 30 = 15000 + 10 \times 100 \times 30$

$= 5000 + 1500$ $= 45000$

$= 6500$

$P = \{10, 100, 5, 30\}$

$$\min\left[\left(P_0 * P_1 * P_2 + P_0 * P_2 * P_3\right), \left(P_1 * P_2 * P_3 + P_0 * P_1 * P_3\right)\right]$$

multiply $P_2 * P_3$

$[1,3]$ ⟨ $[1,1] + [2,3] + P_0 P_1 P_3 \rightarrow 35,000$; $[1,2] + [3,3] + P_0 P_2 P_3 \rightarrow 65,000$ ⟩ chak cost

$$m[i,j] = \begin{cases} 0, & \text{if } i = j \\ \min\{m[i,k] + m[k+1,j] + P_{i-1} P_k P_j, & i < j \\ i \le k < j \end{cases}$$

For

$m[2,5] = \min$ { (1) $m[2,2] + m[3,5] + P_1 P_2 P_5$; (2) $m[2,3] + m[4,5] + P_1 P_3 P_5$; (3) $m[2,4] + m[5,5] + P_1 P_4 P_5$ }

1 return after this

M_1 : 5×10
M_2 : 10×4
M_3 : 4×6
M_4 : 6×10
M_5 : 10×2.

$P = \{5, 10, 4, 6, 10, 2\}$

(i, j)	1	2	3	4	5
1	0	200	320	620	348
2		0	240	640	248
3			0	240	128
4				0	120
5					0

ans

$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + P_0 P_1 P_3 = \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{cases}$$

$$200 + 120 = 320$$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + P_1 P_2 P_4 = 640 \\ m[2,3] + m[4,4] + P_1 P_3 P_4 = \end{cases}$$

$$m[3,5] = \min \begin{cases} m[3,3] + m[4,5] + P_2 P_3 P_5 = \\ m[3,4] + m[4,5] + P_2 P_4 P_5 = \end{cases}$$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 = 1140 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 = 640 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 = 620 \end{cases}$$

MCO(P)
$n \leftarrow length[P] - 1$
for $i \leftarrow 1$ to n
do $m[i,j] \leftarrow 0$

for $l \leftarrow 2$ to n
do for $i \leftarrow 1$ to $n - l + 1$
~~do for $j \leftarrow i + l - 1$~~
do $j \leftarrow i + l - 1$
$m[i,j] \leftarrow \infty$
for $k \leftarrow i$ to $j - 1$
do $q \leftarrow m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$
if $q < m[i,j]$
then $m[i,j] \leftarrow q$
$s[i,j] \leftarrow k$

↓
to print sequence

It assumes that matrix A_i has dimensions
$P_{i-1} \times P_i$ for $i = 1, 2 \ldots n$
The input is a sequence $P = \langle P_0, P_1, \ldots P_n \rangle$

POP($S, 1, n$)
if $i = j$
print ~~print~~ "A"i
else print "("
POP($S, i, S[i,j]$)
POP($S, S[i,j]+1, j$)
Print ")"

→ Tsk

Edit Distance
Bellman Ford
Optimal BST

DFS manner ... BFS manner

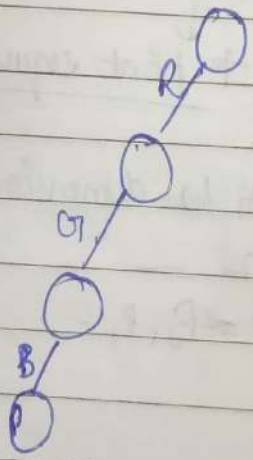
BACKTRACKING & BRANCH & BOUND

Uses Brute Force Approach, which will explore all possible solutions & pick up the desired solutions.

Backtracking is used when you have multiple solutions & you want all those solutions.

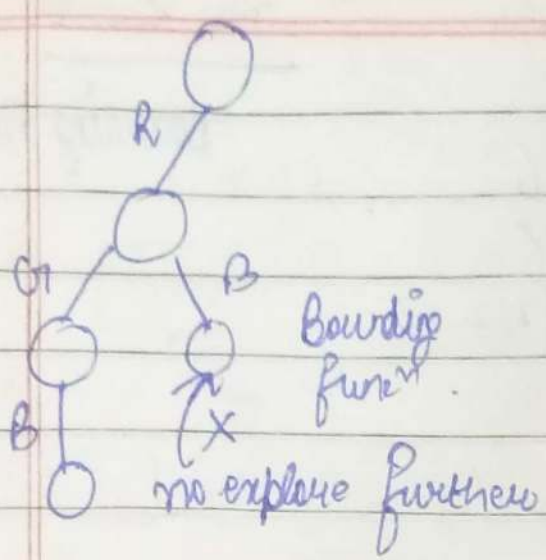
R n B ? want all possible ~~ways~~ ways by which I can colour (one after another).

n colours
⇒ ways = n!
⇒ State Space Tree.



Backtracking problems usually have some constraints & we want the ~~best~~ solution which would satisfy the constraint.

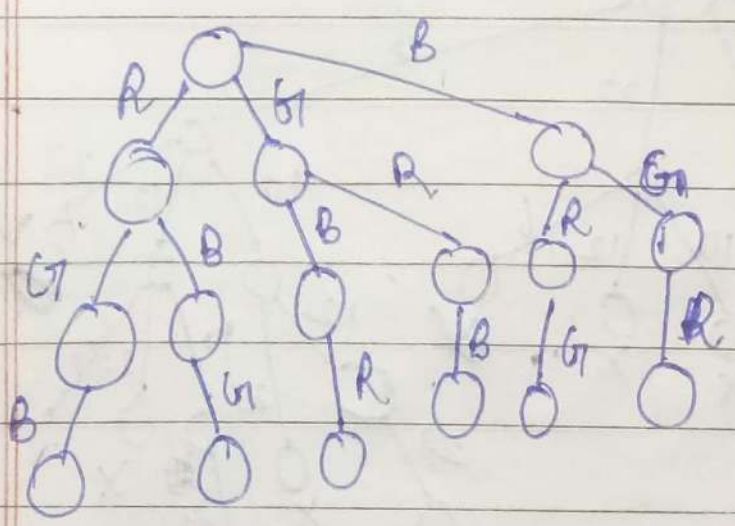
constraint → (B should not be in ~~bet~~ b/w the 2 colours)



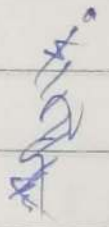
- N Queen
- Sum of subset Problem
- Graph coloring problem

$O(2^n)$.

~~Sum of subset Problem~~



⇒ State space Tree



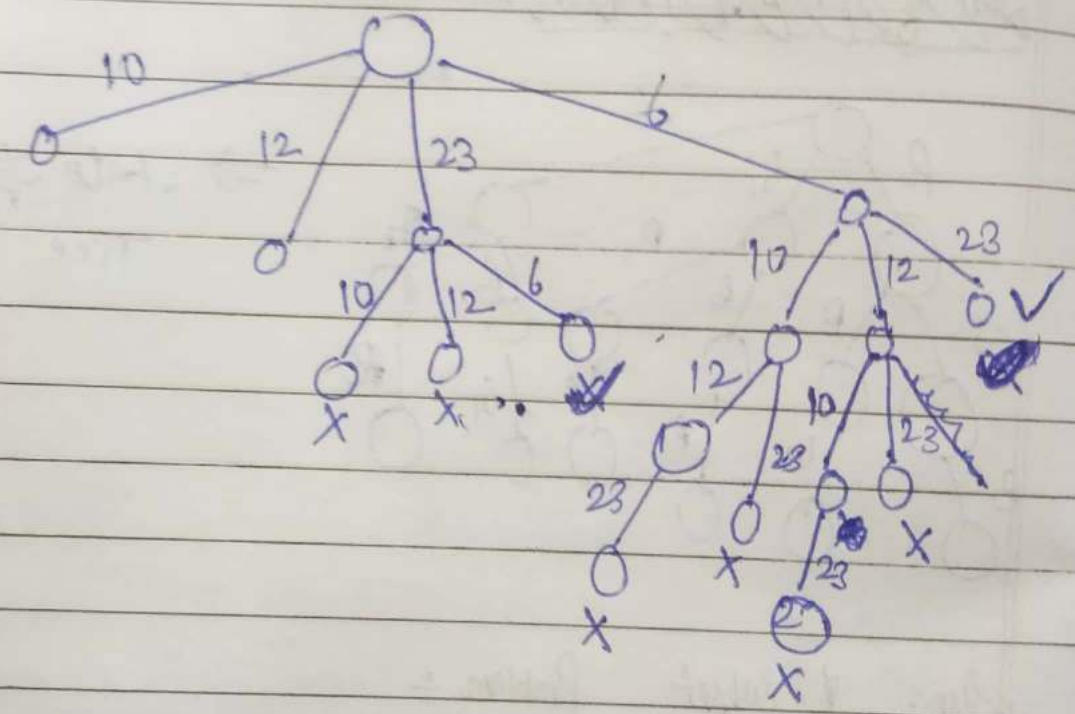
Sum of Subset Problem :-

(10, 12, 23, 6) & m = 29.

(10, 12, 23, 6)

m <= 0

Bounding Function



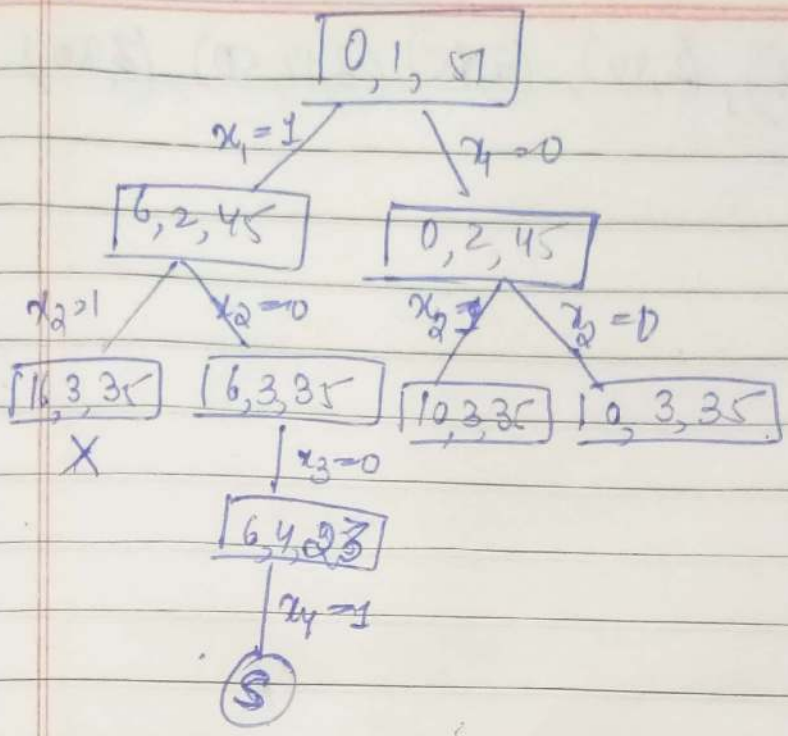
Arrange the elements in non-decreasing order,

(6, 10, 12, 23)

Horowitz

↓ sum of subset Prob (Algo)

x_i denotes either we take or do not take ele.



20 16 50 19

(2, 40), (3.14, 50), (1.98, 100), (5, 95), 1.86 * 20

8.14

(3, 30)

10

10

50 + 95 + 37.2

~~(1.98, 100), (5, 95),~~

↓

(1.98, 100), (3, 30), (5, 95), (3.14, 50),

(5, 95), (2, 40), (1.98, 100)

6.14

30 + 50 + 95

175

11.14

5.86 * 19

111.34

191.34

W = 10