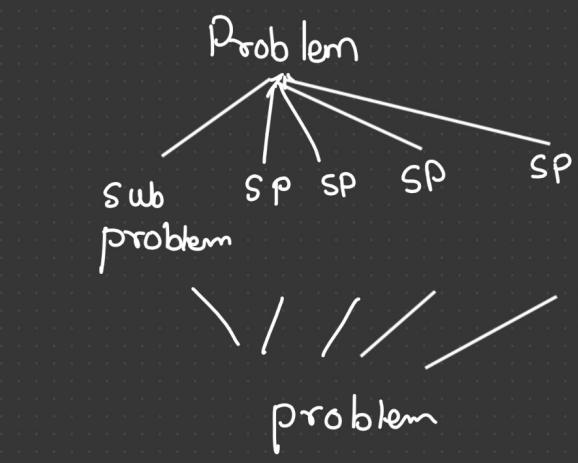
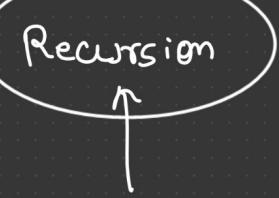


Dynamic Programming



```

int fib ( int n ) { ①
  if (n == 1 || n== 2)
    return 1;
  else
    return fib(n-1)+fib(n-2);
}
  
```



fibonacci Series :-

1, 1, 2, 3, 5, 8, 13, ...

$$\begin{aligned} \text{fib}(n) &= \text{fib}(n-1) + \text{fib}(n-2), n > 2 \\ \text{f}(1) &= 1 \\ \text{f}(2) &= 1 \end{aligned}$$

$$\begin{aligned} \text{f}(3) &= \text{fib}(2) + \text{fib}(1) = 1 + 1 = 2 \\ \text{f}(4) &\dots \end{aligned}$$

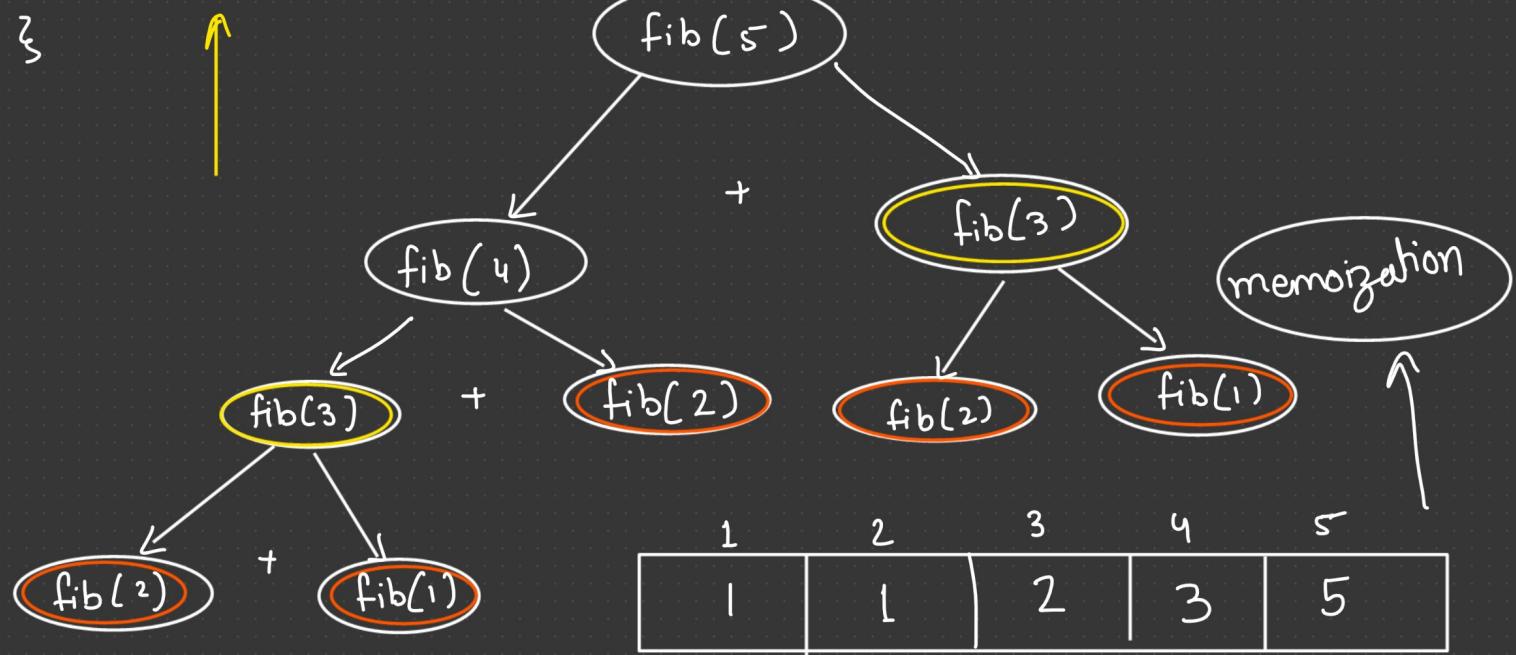
fib(5)

+

fib(3)

memoization

1	2	3	4	5
1	1	2	3	5



```
int f[n+1]; f[1]=f[2]=1;
```

```
int fib ( int n)
```

```
{ if (n == 1 || n == 2)
```

```
    return 1;
```

```
for ( int i=3 ; i<=n ; i++)
```

```
{
```

```
    f[i] = f[i-1] + f[i-2];
```

```
}
```

```
return f[n];
```

```
}
```



Final Answer

```
int fib ( int n) ②
```

```
{ if (n == 1 || n == 2)
```

```
    return 1;
```

```
if [ f[n] != 0 )  
    return f[n];
```

$\rightarrow O(n)$

DP = Recursion + Memoization

1	2	3	4	5
0	0	2	3	5

$\hookrightarrow O(n)$

```
return f[n] = fib(n-1)+fib(n-2);
```

```
}
```

fib(5)

fib(3)=2

fib(1)

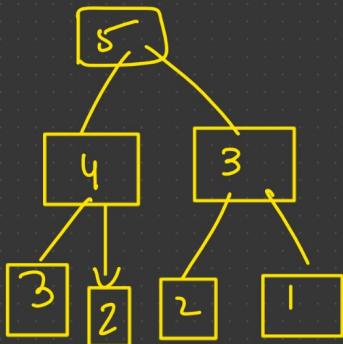
fib(4)

fib(2)=1

2+1

fib(3)

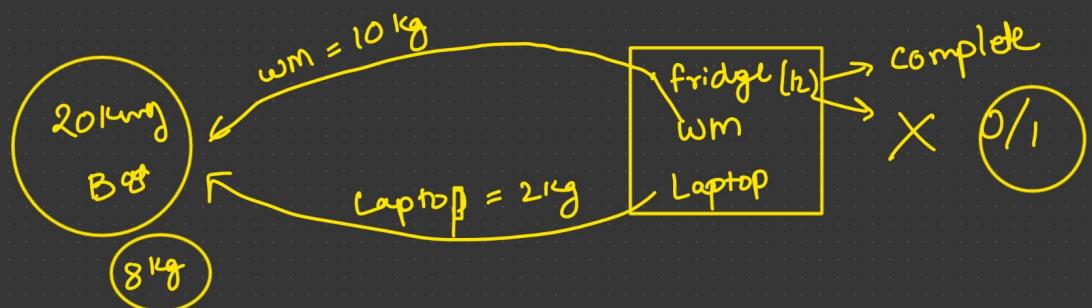
2 = fib(3)



0/1 Knapsack problem

Knapsack problem

- 1) Fractional Knapsack Problem
(Greedy Algorithm)
- 2) 0/1 Knapsack Problem → DP
- 3) Unbounded Knapsack problem.



0/1 Knapsack Problem

$\omega_t [] =$

1	3	4	5
---	---	---	---

Value[] =

1	4	5	7
---	---	---	---

~~per price~~



Greedy Approach failed

$$\omega = 7 \text{ kg} - 5 \text{ kg} = 2 \text{ kg} - 1 \text{ kg} = 1 \text{ kg}$$

$$\text{price} = 71 - +1 = (81)$$

$$\omega = 7 \text{ kg} - 3 - 4 = 0 \quad \xrightarrow{\text{max}}$$

$$\text{Price} = 4 + 5 = \underline{\underline{91}}$$

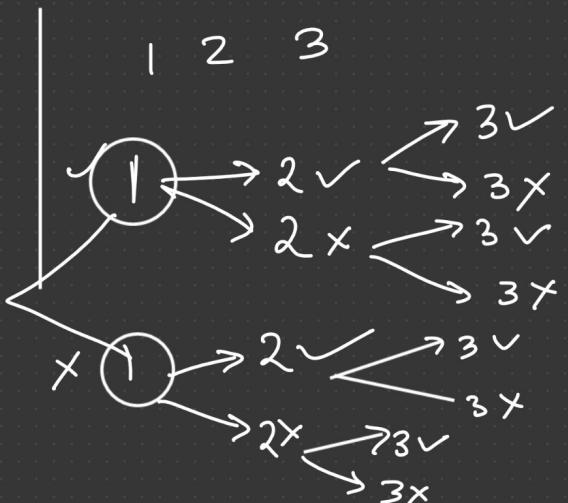
1, 2

1 → 2 ✓

1 → 2 ×

X | → 2 ✓

$$x_1 \rightarrow 2x$$

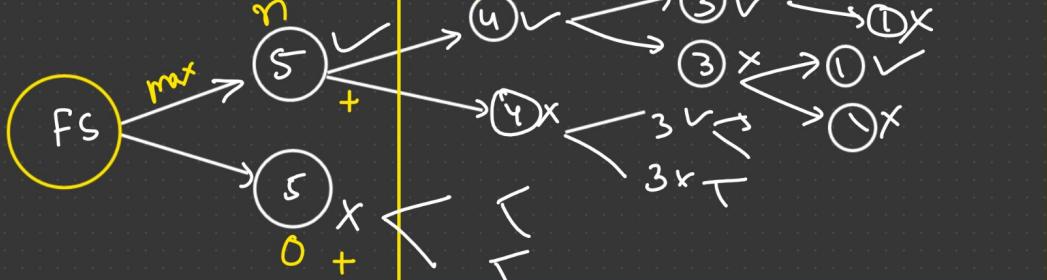


It is growing exponentially

$$\boxed{1, 3, 4,} \Big| 5$$

$\rightarrow n \rightarrow$ Total element = 4

$\rightarrow n-1 \rightarrow \text{index} = 3$



Given:-

```
int wt[] = { 1, 3, 4, 5 };
int value[] = { 1, 4, 5, 7 };
int w = 7;
int n = 4;
```

```
int knapsack( int wt[], int value[], int w, int n )
```

{

// Base condition.

```
if ( n == 0 || w == 0 )
    return 0;
```

```
if ( wt[n-1] <= w )
{
```

```
    return max [ value[n-1] + knapsack( wt, value, w-wt[n-1], n-1 ),
                  0 + knapsack( wt, value, w, n-1 ) ];
}
```

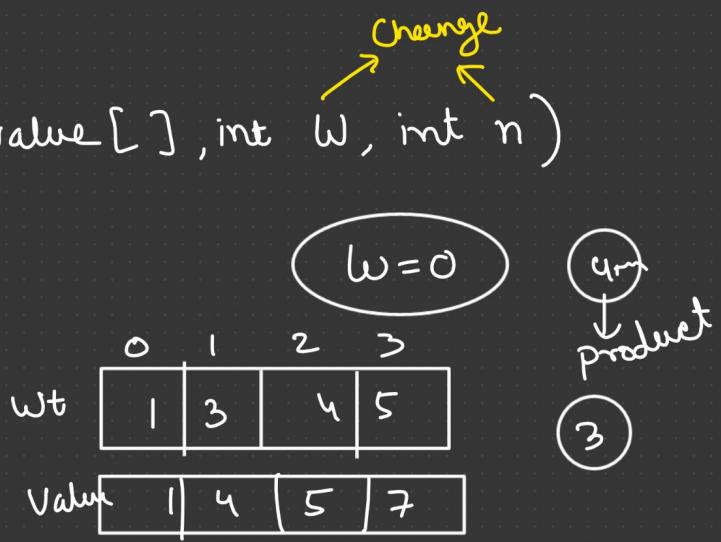
}

else

{

```
    return knapsack( wt, value, w, n-1 );
}
```

}



Using memoization

```
int wt[] = { };  
int value[] = { };  
int dp[n+1][w+1] = { -1 };
```

```
int knapsack( int w, int n )
```

```
{
```

```
if ( n == 0 || w == 0 )  
    return 0;
```



```
if ( dp[n][w] != -1 )  
    return dp[n][w];
```

```
if ( wt[n-1] <= w )
```

```
{
```

```
return dp[n][w] = max { value[n-1] + Knapsack( w - wt[n-1], n-1 ),  
                      0 + Knapsack( w, n-1 ) };
```

```
 }
```

```
else  
    return dp[n][w] = knapsack( w, n-1 );
```

```
}  
Return dp[n][w]; → Final price.
```

$O(n^2)$

3 → n (which item I want check)

3 → wt (weight of n^{th} element)

3 → price (final price)

	0	1	2	3	4	5	6	7	wt of Bag (w)
P	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	1	1	
2	2	1	1	4	8				\downarrow w
3	3								\downarrow n
4	4								
5									
6									
7									

\downarrow w

	0	1	2	3	4	5	6	7	wt of Bag (w)
P	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	1	1	
2	2	1	1	4	8				\downarrow w
3	3								\downarrow n
4	4								
5									
6									
7									

\downarrow n

Ans = $dp[n][w]$

item no (n)

.

Top-Down
approach

	0	1	2	3	4	$\rightarrow w[i]$
0	0	0	0	0	0	
1	0	4	4	4	4	
2	0	4	4	7	7	$\leftarrow i-1$
3	0	4	4	7	7	$\leftarrow i$

$$n \downarrow \quad \text{KS}(\overset{\omega}{4}, \overset{n}{3})$$

$$dp[i][\omega] = dp[i-1][\omega]$$

w^{++}

```

for( i=0 ; i<=n ; i++ )
{
    for( w=0 ; w<=w ; w++ )
}

```

3

```
int knapsack ( int W, int n )
```

$$\text{int } dp[n+1][\omega+1] = \{-1\};$$

```
{  
    for (int i=0 ; i <= n ; i++)  
    {  
        for (int w=0; w <= W; w++)  
        {  
            ...  
        }  
    }  
}
```

else if ($\text{wt}[i-1] \leq \omega$)

$$dp[i][\omega] = \max \left\{ \begin{array}{l} \frac{\text{value}[i-1] + dp[i-1][\omega - \omega_{t[i-1]}]}{0 + dp[i-1][\omega]} \\ \end{array} \right\}$$

else $d[i][\omega] = dp[i-1][\omega];$

} return[n][w];

0	1	2
1	2	3

4	3	2
value		

$$w = 4, \max = 7 / -$$

```

    for (i=0; i<=n; i++)
    {
        for (w=0; w<=W; w++)
    }
}

```

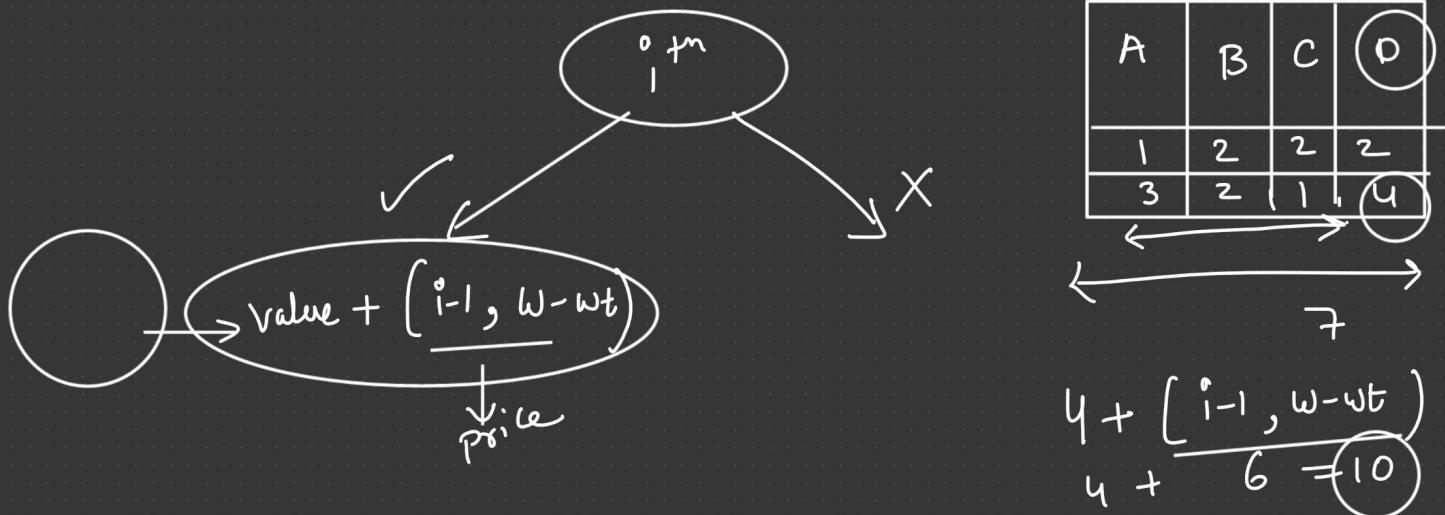
o o o o o o

```

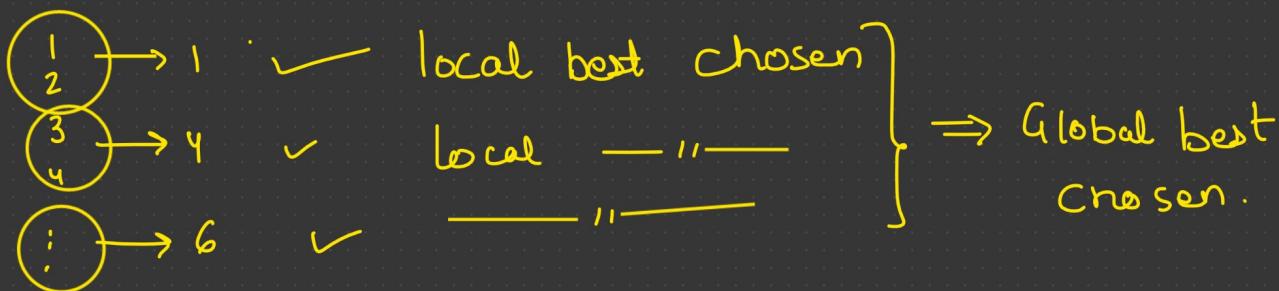
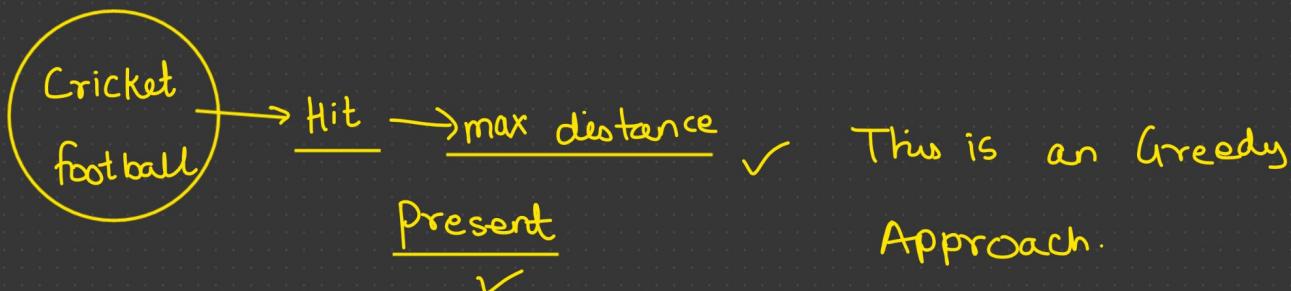
for( int i=0 ; i<=n ; i++)
    dp[i][0] = 0;

for( int j=0 ; j<=w ; j++)
    dp[0][j] = 0;

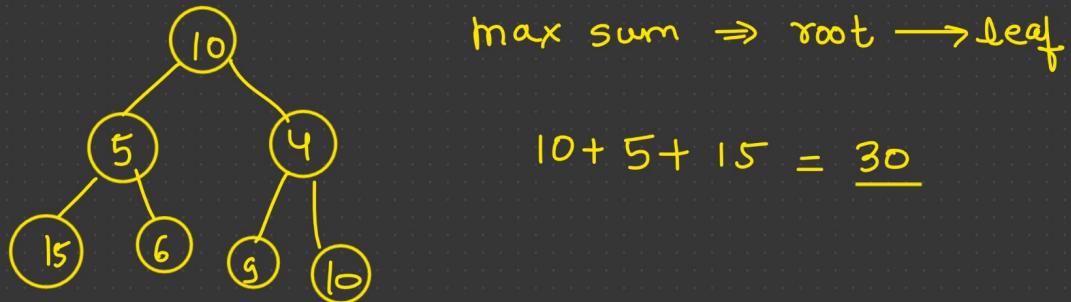
```



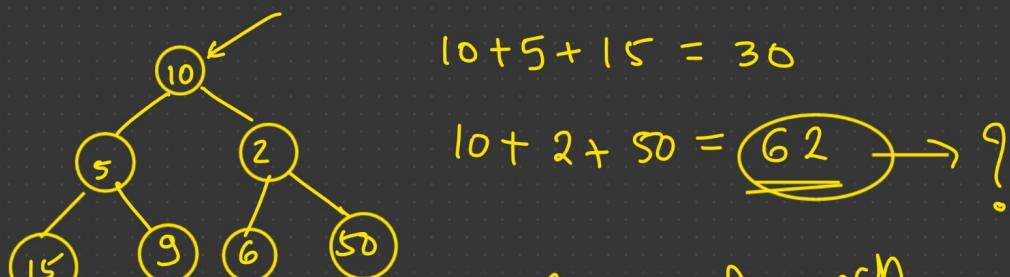
Greedy Algorithm



Problem :-



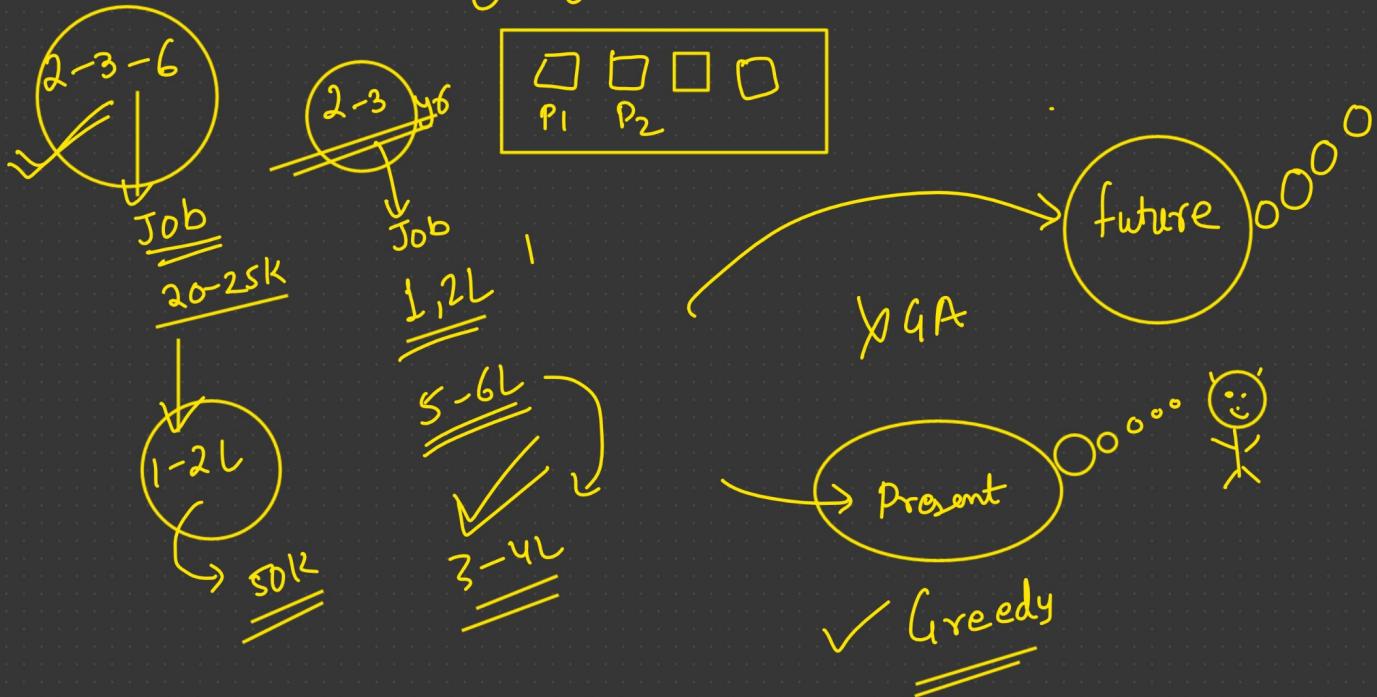
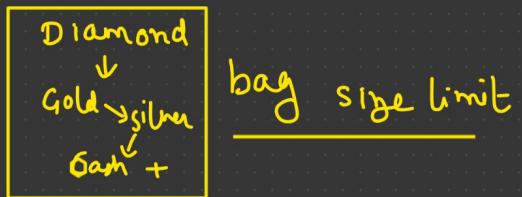
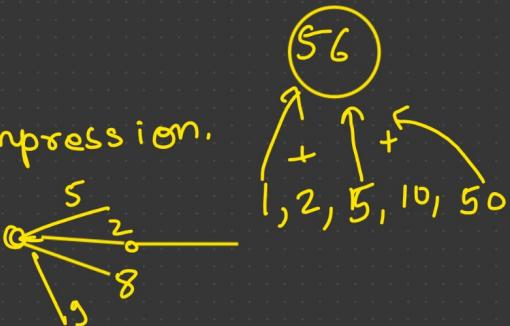
Advantage & Disadvantage :-



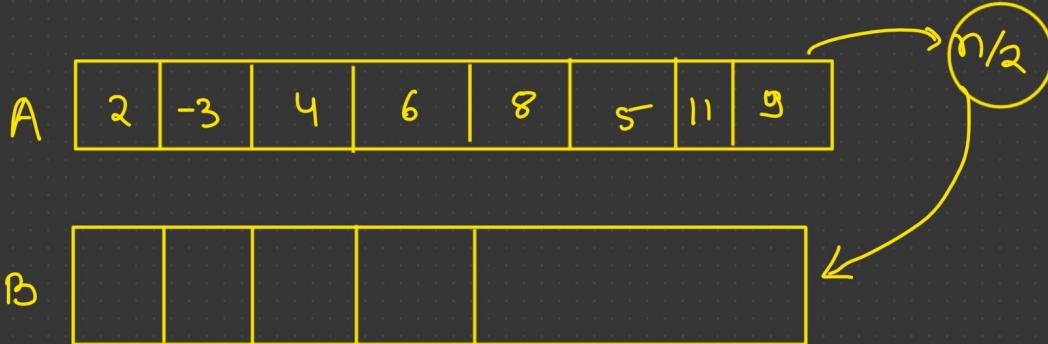
Greedy Approach failed

Ex :-

- 1) Sorting \rightarrow Selection sort , Topological sort.
- 2) Priority Queue \rightarrow Max Heap , min Heap
- 3) Huffman Encoding \rightarrow Encryption , Compression.
- 4) Prim's , Kruskal's , Dijkstra's
- 5) coin change problem
- 6) Fractional knapsack Problem .
- 7) Job Scheduling algorithm.



Problem: You are given an array A ,of n elements. You have to remove exactly $n/2$ elements from an array and add it to another array B (initially empty).Find the maximum and minimum values of difference between these two arrays. The difference between those two arrays is $\text{sum}(\text{abs}(A[i]-B[i]))$.



$$\sum_{i=0}^{n/2} |A[i] - B[i]| \rightarrow \max$$

$\downarrow \min$

Ex:- $2, -3, 0, 10, 12, 5$ n=6

0	1	2
12	10	5

0 1 2

0	1	2
-3	0	2

0 1 2

$$(A[0]-B[0]) + (A[1]-B[1]) + (A[2]-B[2])$$

$$\sum_{i=0}^{n/2-1} |(A[i] - B[i])|$$

$$(2-0) + (-3-10) + (12-5) = 2 + 13 + 7 = 22$$

$$(12-(-3)) + (10-0) + (5-2) = 15 + 10 + 3 = 28 \rightarrow \text{maximum}$$

0	1	2
-3	2	10

0 1 2

$$\Rightarrow |(-3-0)| + |(2-5)| + |10-12|$$

0	1	2
0	5	12

0 1 2

$$3 + 3 + 2 = 8 \rightarrow \text{minimum.}$$

$$\text{Maximum Sum} = (a_0 - b_0) + (a_1 - b_1) + (a_2 - b_2)$$

\downarrow
 max + \downarrow
 max + \downarrow
 max

$$A = \boxed{12, 10, 5}, \quad B = \boxed{-3, 0, 2}$$

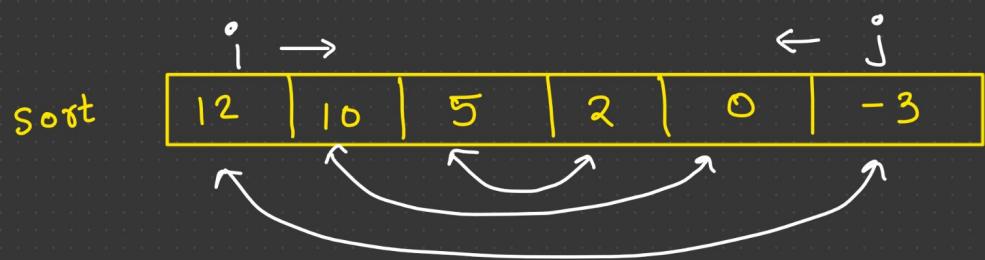
Step1 :- Sort all the numbers in descending order.

Step2 :- $A = [0 - n/2]$

$$B = [n-1, \dots, n/2+1]$$

Step3 :- Find $\sum |a_i - b_i|$. This will give me maximum sum.

Alash + Lalach = New Approach.



```
vector<int> a(n);
```

```
for (int i=0; i<n; i++)
  cin >> a[i];
```

```
sort(a.begin(), a.end());
```

```
int j = n-1;
```

```

int max = 0, min = 0;
for (i=0; i<n/2; i++)
{
  max = max + abs(a[i] - a[j]);
  min = min + abs(a[2*i] - a[2*i+1]);
  j--;
}
cout << max << " " << min;
  
```

$$\text{Minimum Sum} := (a_0 - b_0) + (a_1 - b_1) + (a_2 - b_2)$$

$$\downarrow \min + \downarrow \min + \downarrow \min = \min$$

0	1	2	3	4	5
12	10	5	2	0	-3

↓ ↓ ↓
 min + min + min = min

$3 - 0 = 3$
 $3 - 1 = 2$
 $3 - 2 = 1$
 $3 - 3 = 0$ ← min

$$|12 - 10| + |5 - 2| + |0 - (-3)|$$

$$2 + 3 + 3 = \underline{\circled{8}} \leftarrow \min$$

C++ program to find minimum number of denominations // Coin change problem.

1, 2, 5, 10, 20, 50, 100, 500, 1000

↓
infinite supply of each coins ↳ find minimum coins required.

$$563 = 500 + 50 + 10 + 2 + 1$$

$$490 = 100 + 100 + 100 + 100 + 50 + 20 + 20$$

if (Amount >= a[i])
 {
 Amount = Amount - a[i];
 }

int main()

{
 int deno[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};
 vector<int> ans; int n = 9; int Amt = 153;
 Sort(deno, deno + n);

for (int i = n - 1; i >= 0; i--) $\rightarrow O(n^2)$

 while (Amt >= deno[i]) $\rightarrow n - n^2$

 {
 Amt = Amt - deno[i];
 ans.push(deno[i]);

}

 if (Amt == 0)
 break;

}

 for (int i = 0; i < ans.size(); i++)
 cout << ans[i] << " ";

}

Activity Selection Problem

Problem -

We are given n activities with their start and finish times. We have to select the maximum number of activities such that no two selected activities overlap.

start	6	5	1	8	3	8
finish	11	7	4	12	6	9

Solution :- 1) Sort all the event on the basis of its finish time.

start	1	3	5	8	6	8
finish	4	6	7	9	11	12

2) Select the event that finish first.
Then Jump to next event.

start	1	3	5	8	6	8
finish	4	6	7	9	11	12

↑

3) If starting time of next event \geq finish time of current event. Then select that event. Otherwise move to next event.

start	1	3	5	8	6	8
finish	4	6	7	9	11	12

3 \neq 4 5 $>$ 4 8 $>$ 7 6 \neq 9 8 \neq 9

\checkmark
 $n = 3$

C++

Struct event

```
{
    int start;
    int end;
};
```

6	5	1	8	3	8
11	7	4	12	6	9

bool compare (event e1, event e2)

```
{
    return (e1.end < e2.end);
```

}

int main()

{

```
Event e[ ] = { {6,11}, {5,7}, {1,4}, {8,12}, {3,6},
                {8,9} };
```

int n = 6;

sort (e, e+n, compare);

return type bool

int current = 0;

cout << e[0].start << " " << e[0].end;

for (int next = 1 ; next < n ; next ++)

{

if (e[current].end <= e[next].start)

{ cout << e[next].start << " " << e[next].end;

current = next;

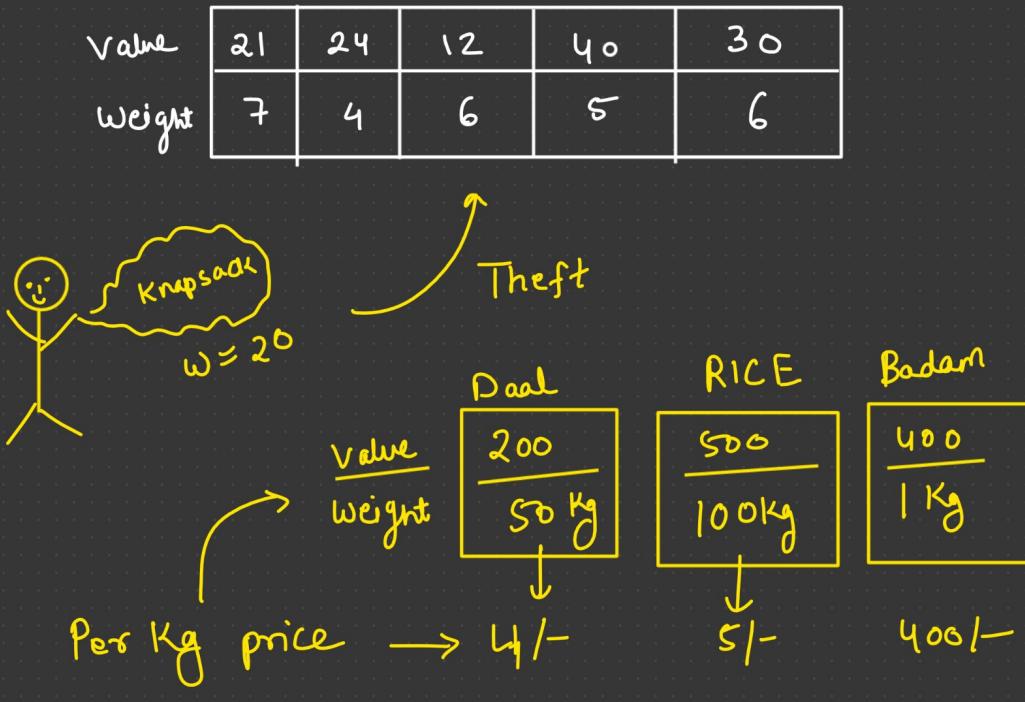
}

}

Fractional Knapsack Problem

Problem

We are given n items with {weight, value} of each item and the capacity of knapsack (bori) W . We need to put these items in the knapsack such that the final value of items in the knapsack is maximum.



Sort → Price ($\uparrow \dots \downarrow$)

Step 1:- Find per kg price of each item

Value	21	24	12	40	30
Weight	7	4	6	5	6
Price	3	6	2	8	5

Step 2:- Sort in descending order on the basis
price

Value	40	24	30	21	12
Weight	5	4	6	7	6
Price	8	6	5	3	2

Step 3 :- Pick from the starting till our Capacity
of Knapsack.

$$W = 20$$

$$\begin{array}{r} -5 \\ \hline 15 \\ -4 \\ \hline 11 \\ -6 \\ \hline 5 \end{array}$$

$$\text{Price} = 40 + 24 + 30 + 15 = 70 + 24 + 15 = \underline{\underline{109}}/-$$

$$\text{Weight} = 5 + 4 + 6 + 5$$

struct Item

{

public:

int value;

int weight;

};

bool prateek(Item I1, Item I2)

{

float P1 = I1.value / I1.weight;

float P2 = I2.value / I2.weight;

return (P1 > P2);

}

```

int main()
{
    Item a[] = { {21,7}, {12,6}, {30,6}, {24,4}, {40,5} };
    int n = 5, wt = 20, amount = 0;
    sort( a, a+n, prateek);

    for( int i=0; i<n ; i++)
    {
        if ( a[i].weight <= wt)
        {
            wt = wt - a[i].weight;
            amount = amount + a[i].value;
        }
        else
        {
            amount = amount + (a[i].value/a[i].weight)*wt;
            break;
        }
    }

    cout << "Final Amount " << amount;
}

```

Optimal Merge Pattern

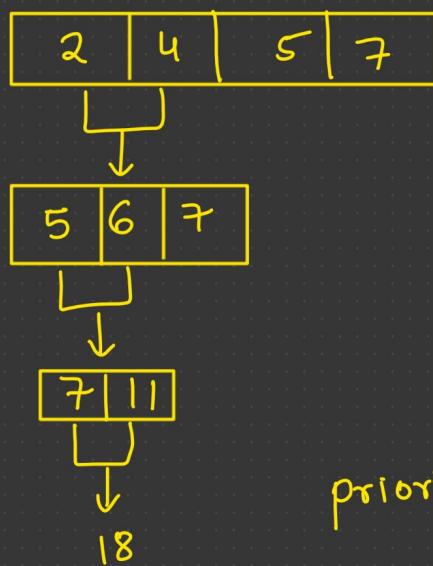
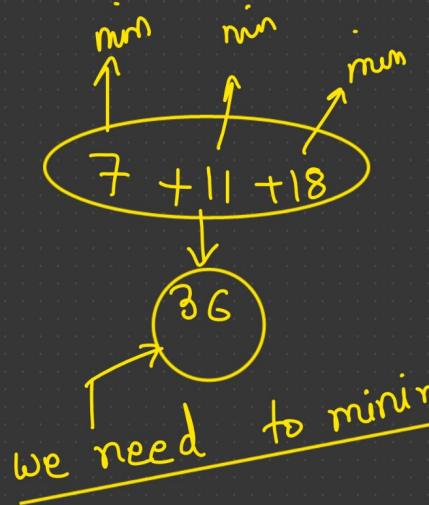
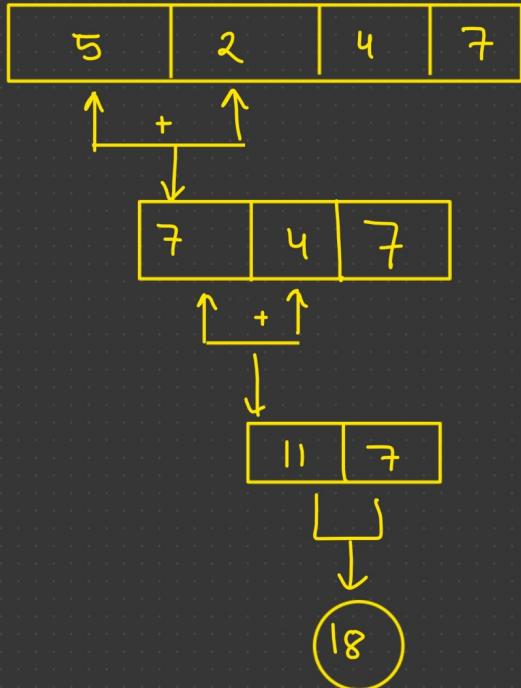
Problem

You are given n files, with their computation times in an Array. You can perform the following operation Operation:

Choose/ take any two files, add their computation times and append it to the list of computation times.
{Cost = Sum of computation times}

Do this until we are left with only one file in the array. We have to do this operation so that we can get the minimum cost finally.

Ex:-



$$\begin{aligned}
 & 6 + 11 + 18 \\
 = & 35
 \end{aligned}$$

priority Queue
 ↓
 min Heap

```

int main()
{
    int time[] = { 4, 5, 2, 7 }; int amount = 0;
    priority_queue <int, vector<int>, greater<int>> pq;

    for (int i=0 ; i<4 ; i++)
        pq.push (time[i]);
    while (pq.size() > 1)
    {
        int x = pq.top();
        pq.pop();
        int y = pq.top();
        pq.pop();
        amount = amount + (x+y);
        pq.push (x+y);
    }
    cout << amount;
}

```

pq

2	4	5	7
↑	↑		
top	top		

5	6	7
---	---	---

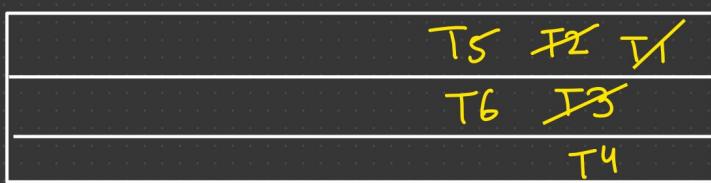
Minimum Number Platform

Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits. We are given two arrays that represent the arrival and departure times of trains that stop.

Arrival	9:00	9:40	9:50	11:00	15:00	18:00
departure	9:10	12:00	11:20	11:30	19:00	20:00

$T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5 \quad T_6$

Railway
station



pq	9:10	11:20	11:30	12:00	15:00
------	------	-------	-------	-------	-------

\uparrow
 3 train

$$\text{No. of platform} = \underline{\underline{3}}$$

```

int main()
{
    int arr[] = {900, 940, 950, 1100, 1500, 1800};
    int dep[] = {910, 1200, 1120, 1130, 1900, 2000};

    vector<pair<int, int>> v(6);

    for (int i=0; i<6; i++)
        v[i] = {arr[i], dep[i]}; second first

    sort(v.begin(), v.end());
}

priority_queue<int, vector<int>, greater<int>> pq;

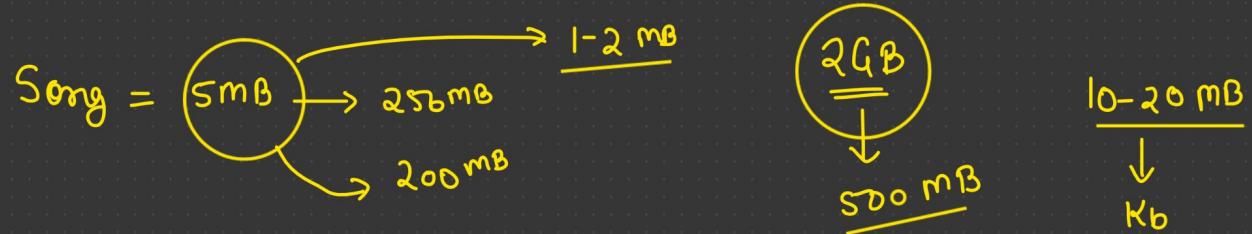
int count = 1;
pq.push(v[0].second); insert departure time of 1st train

for (int i=1; i<n; i++)
{
    if (pq.top() >= v[i].first)
    {
        count++;
        pq.push(v[i].second);
    }
    else
    {
        pq.pop();
        pq.push(v[i].second);
    }
}
cout << count;

```

Efficient Huffman Coding for Sorted Input.

file = 100 MB \longrightarrow 40 MB, 60 MB
Compress
+ encrypt



My name is prateek Jain.

1 char = 1 byte

$$= 2^5 = 2^5 \text{ bytes} - 6 \text{ byte} = 19 \text{ bytes}$$

a - 3
e - 3
i - 2
j - 1
m - 2
n - 2
p - 1
r - 1
t - 1
k - 1
s - 1
y - 1

sort according to frequency
→

j	p	r	t	k	s	y	i	m	n	a	e
1	1	1	1	1	1	1	2	2	2	3	3

5-bit

8-bit

101

5 bit × 3 = 15 bit

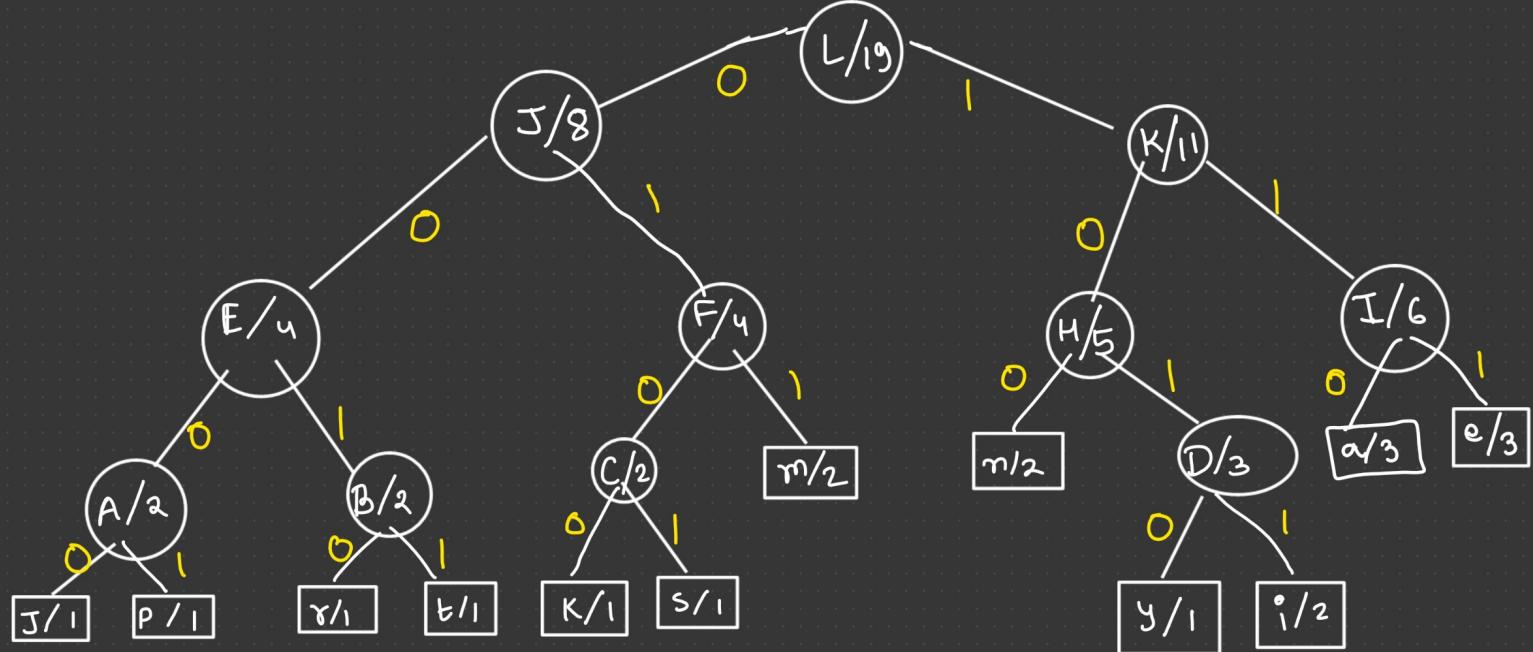
1 bit × 3 = 3-bit

4 byte → 1 byte

j	p	r	t	k	s	y	i	m	n	a	e
1	1	1	1	1	1	1	2	2	2	3	3

How code is generated

A/2, B/2, C/3, D/4, E/4, F/4, G/5, H/6, I/8, J/11



$$\begin{array}{l|l|l}
 J = 0000 & K = 0100 & y = 1010 \\
 p = 0001 & S = 0101 & i = 1011 \\
 r = 0010 & m = 011 & a = 110 \\
 t = 0011 & n = 100 & e = 111
 \end{array}$$

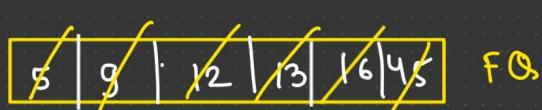
$$\begin{aligned}
 &= 7 \times 4 + 4 \times 3 + 2 \times 4 + 6 \times 3 = 28 + 12 + 8 + 18 \\
 &= 40 + 26 = \frac{66}{8} \text{ bits} \\
 &\Rightarrow 8 \text{ byte } 2 \text{ bits}
 \end{aligned}$$

Implementing Huffman coding using 2 queue:-

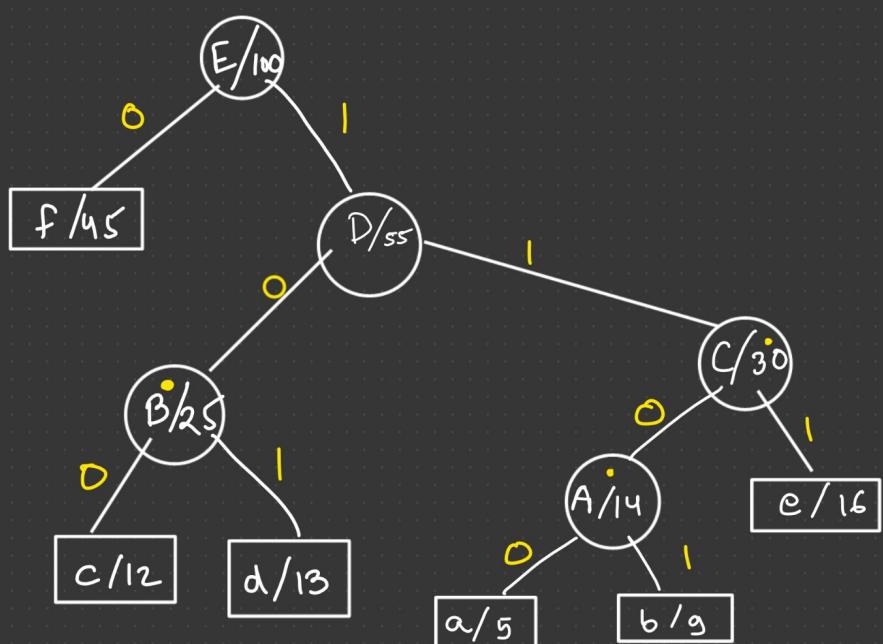
a	b	c	d	e	f
5	9	12	13	16	45

unsorted = $O(n \log n)$

sorted = $O(n)$

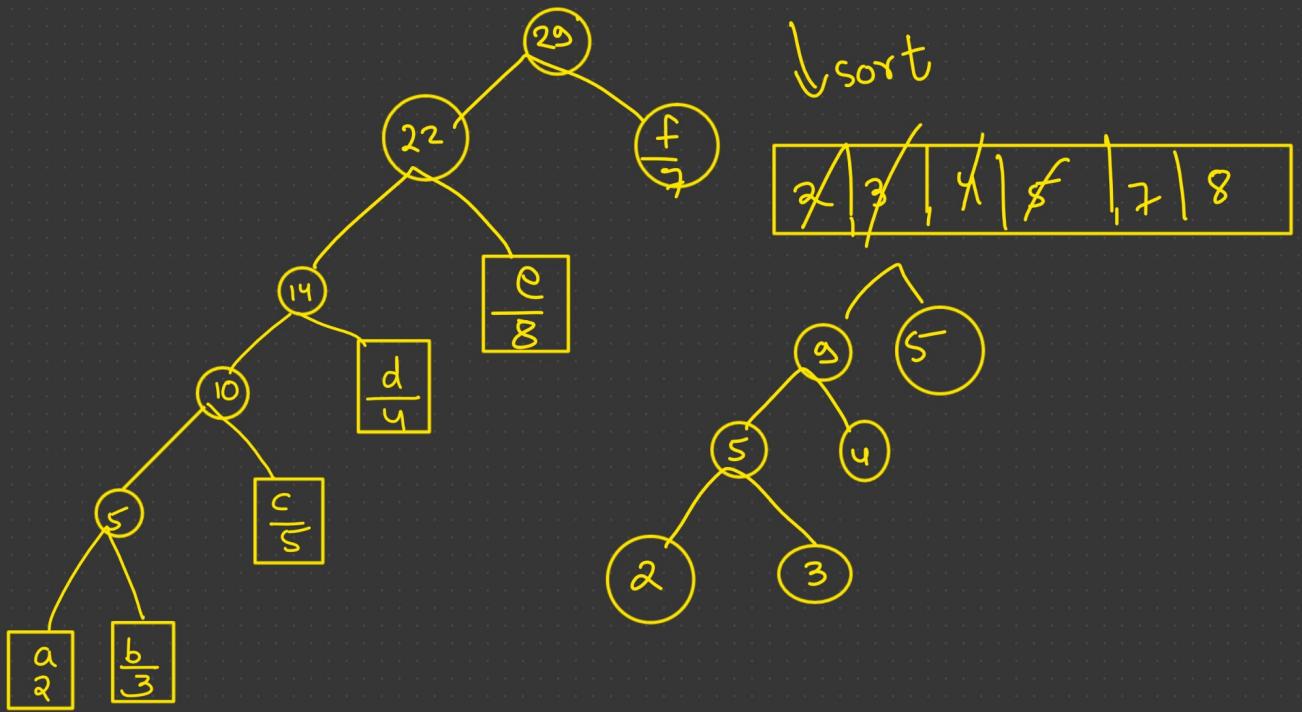


$a - 1100$
 $b - 1101$
 $c - 100$
 $d - 101$
 $e - 111$
 $f - 0$



Given are N ropes of different lengths, the task is to connect these ropes into one rope with minimum cost, such that the cost to connect two ropes is equal to the sum of their lengths.

a	b	c	d	e	f	
2	3	5	4	8	7	= 29



In a candy store, there are N different types of candies available and the prices of all the N different types of candies are provided. There is also an attractive offer by the candy store. We can buy a single candy from the store and get at most K other candies (all are different types) for free.

Find the minimum amount of money we have to spend to buy all the N different candies.
 Find the maximum amount of money we have to spend to buy all the N different candies.

Input :

price[] = {3, 2, 1, 4}

$k = 2$

Output :

Min = 3, Max = 7

a	b	c	d
3	2	1	4

$$K = 2$$

↓ Sort

1	2	3	4
---	---	---	---

$$1 + 2 = 3 = \text{min}$$

1	2	3	4
---	---	---	---

$$4 + 3 = 7 = \text{max}$$

```
for(i=0; i<n; i++)
{
    min price = min price + a[i];
}
n = n - K;
```

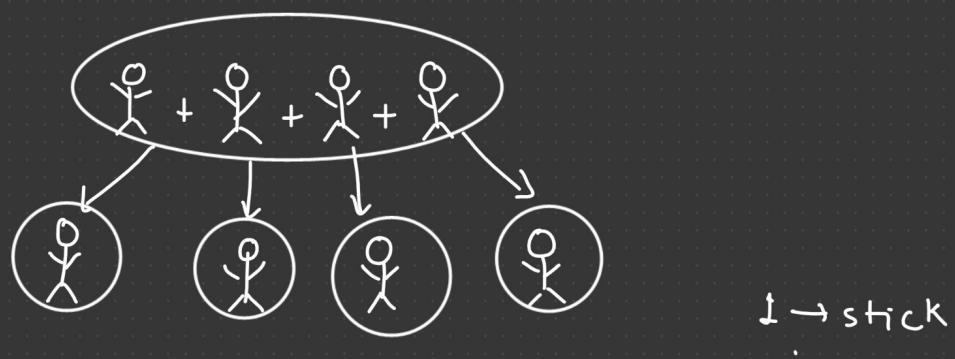
```
int l = 0;
for(i=n-1; i >= l; i--)
{
    max price = max price + a[i];
}
l = l + K;
```

Divide & Conquer

1) Binary Search

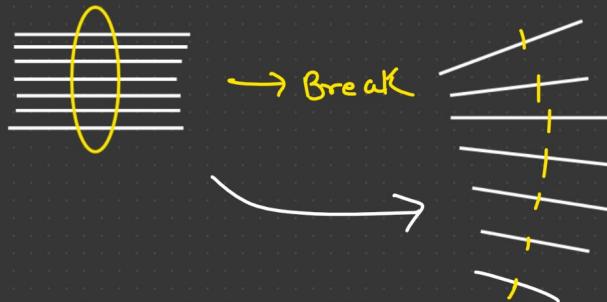
2) Merge Sort + Recursion

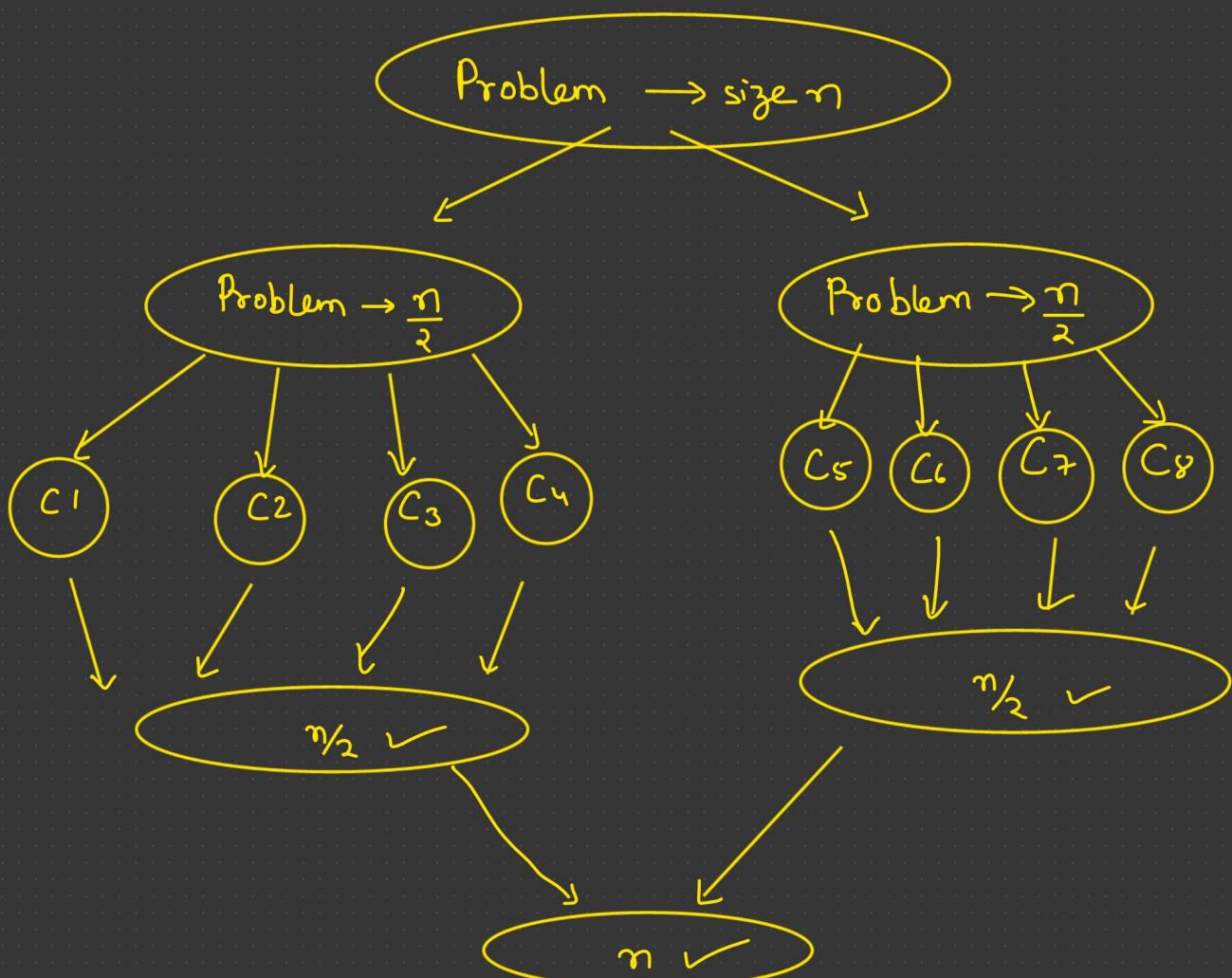
3) Quick Sort



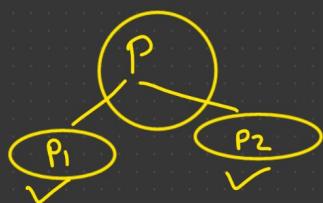
Father \rightarrow 7 son's

Property



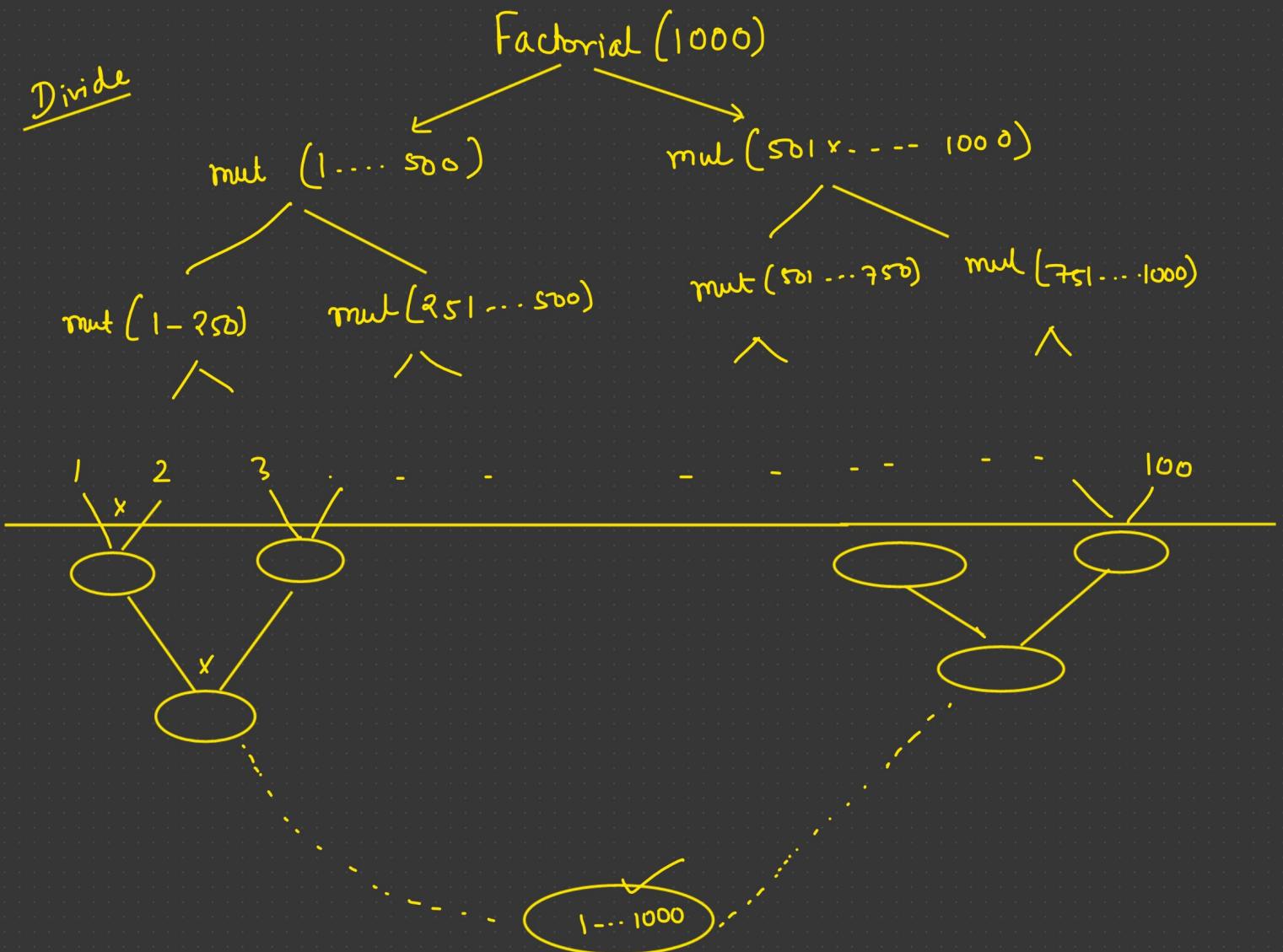


Problem → D & C ✓
→ D & C ✗



Bigger Problem → small problem 1
→ _____ 2
→ _____ 3
→ _____ 4

- Each smaller problem must be independent of each other.
- Each smaller problem must be similar to the bigger problem.



D&C Strategy ?

- 1) **Divide** — Breaking the problem into sub problems that are themselves smaller instance of the same type of problem.
- 2) **Recursion** — Recursively solving these sub-problems.
- 3) **Conquer** — Appropriately combining these results.

DandC(P)

{

if(small(P))

return Solution(P);

// Divide the problem P into Subproblem (P_1, P_2, \dots, P_k)

return Combine(DandC(P_1),

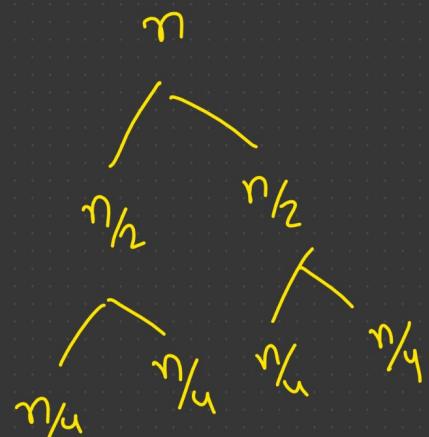
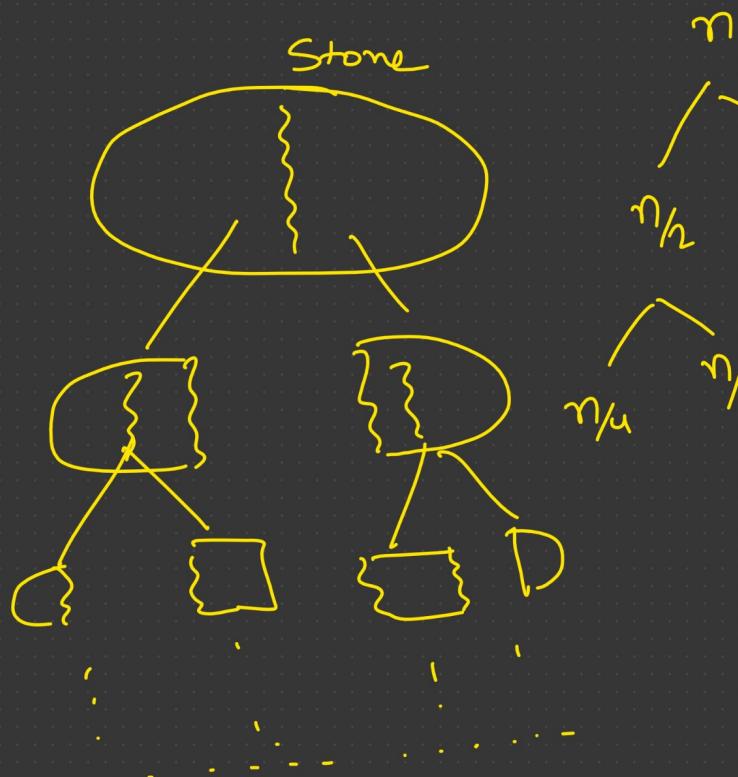
DandC(P_2),

DandC(P_3),

:

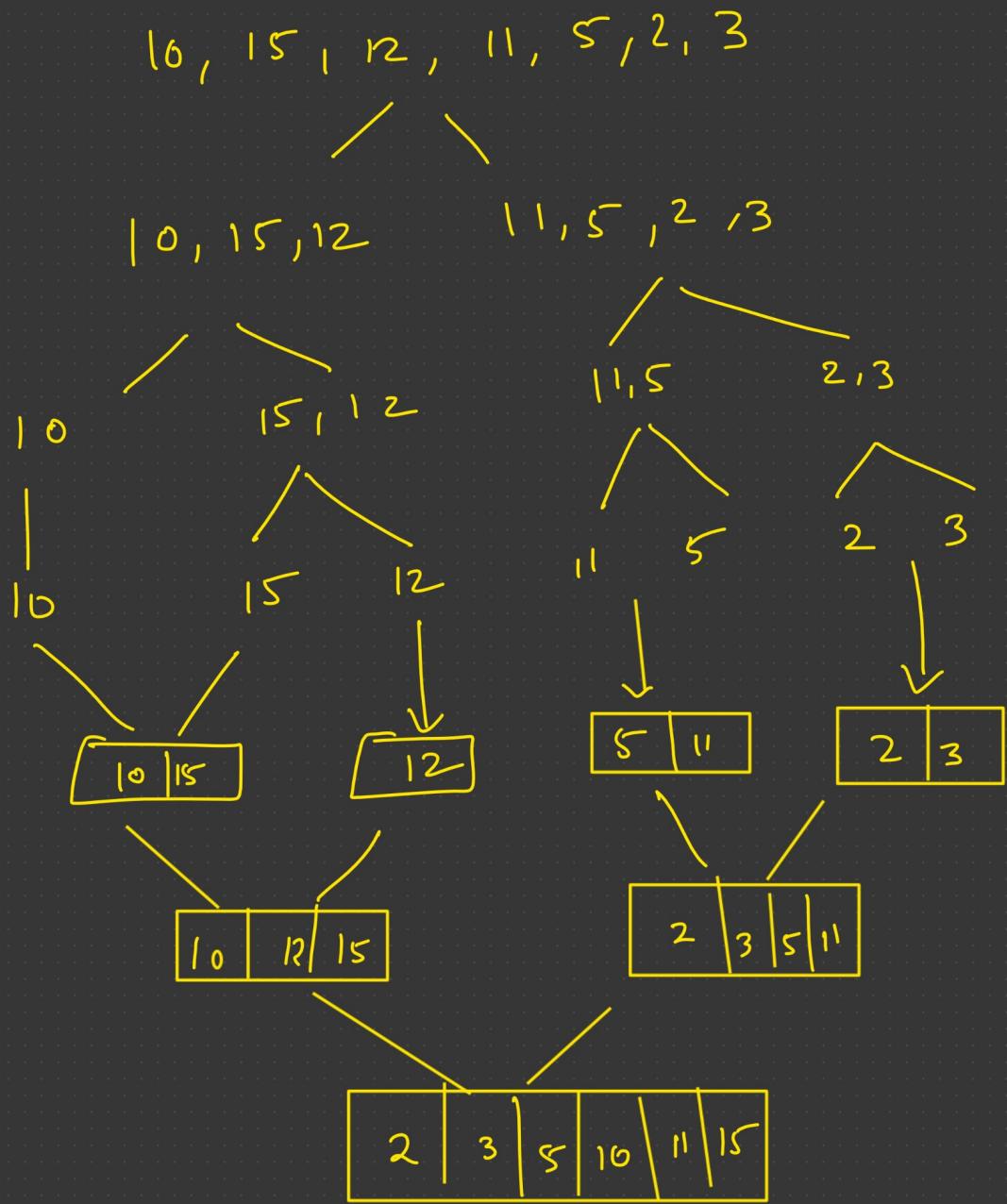
DandC(P_k));

}

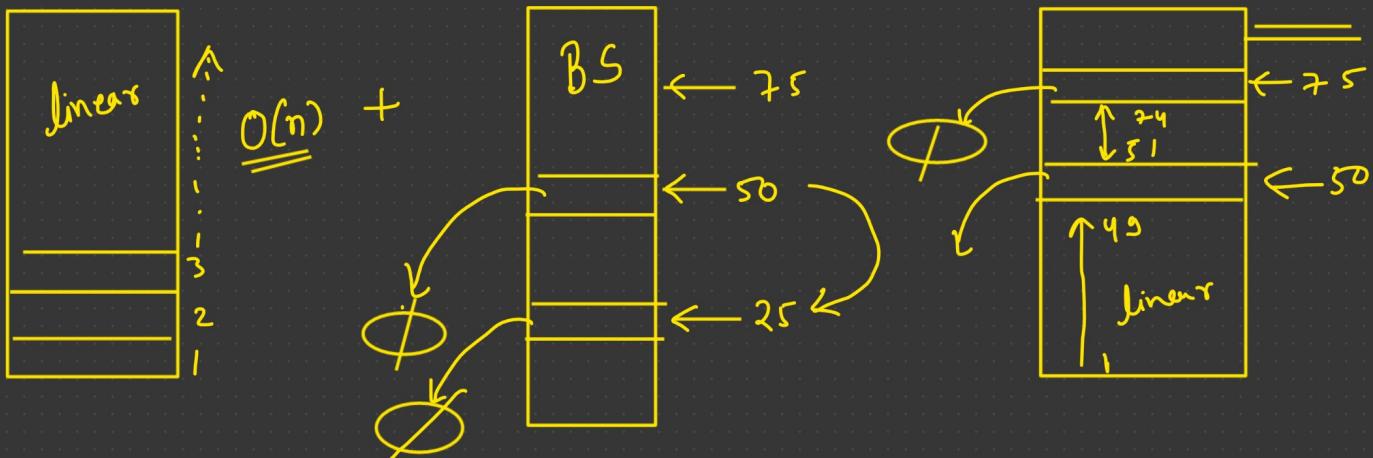


10, 15, 12, 11, 5, 2, 3

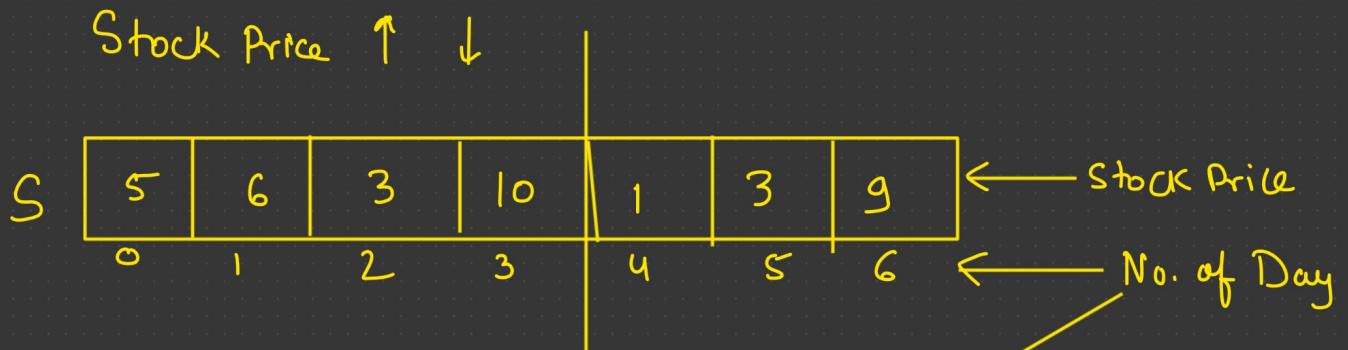




Ex:- You are given 2 eggs & 100 story building.



Maximize the profit for stock market :-



$i \rightarrow$ stock Buy

$i < j$

$j \rightarrow$ stock sell

$$\text{Profit} = s[j] - s[i] = \text{S.P} - \text{C.P}$$

1) Bruteforce :-

```
for (i=0 ; i<n-1 ; i++)
```

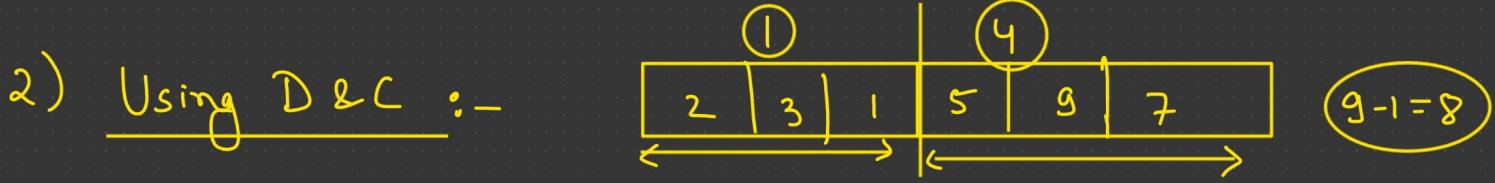
```
{   for(j=i+1 ; j < n ; j++)
```

```
{
```

```
    if ( Profit < s[j] - s[i] )
```

```
        Profit = s[j] - s[i]
```

```
}
```



a) left array $\text{MaxStock}(S, \text{left}, \text{mid}) = \text{PL} = 1$

b) Right array $\text{MaxStock}(S, \text{mid}+1, \text{right}) = \text{PR} = 4$

c) $\text{PM} = \max_{g} [\text{Array}[mid+1, right]] - \min_{l} [\text{Array}[left, mid]]$
 $g - l = 8$

$$\begin{aligned}\text{Profit} &= \max(\text{PM}, \text{PL}, \text{PR}) \\ &= \max(8, 1, 4) = 8 \rightarrow \underline{\underline{O(n \log n)}}\end{aligned}$$

DP $\rightarrow \underline{\underline{O(n)}}$

Maximum SubArray Subsequence :-

$\underbrace{-5, 6, -2}_{i=2}, \underbrace{11, -14, -5, 1}_{i < n}$

$O(n \log n) \rightarrow \underline{\underline{D \& C}}$

$\max = \begin{matrix} -5 \\ + \\ 10 \\ 15 \end{matrix}$

```
for (i=0; i<n; i++)
{
    for(j=i+1; j<n; j++)
        {
             $\max$ 
        }
}
```

$\rightarrow \underline{\underline{O(n^2)}}$

String Algorithms

gm
gmail.com

→ Auto-completion.

Prateek

	<u>Prefix</u>
	P
	Pr
	Prat
	Prate
	Pratee
	Prateek
	.

Suffix

K
ek
eeK
teek
ateek
rateeek
Prateek

Compare (Prefix, Text list)
{ return list
}

(Prk)
X
(Ptk) X

Text = "My name is Prateek" → string matching

Pattern = "Prateek"

- 1) Brute force approach →
- 2) Robin Karp
- 3) KMP

Brute force Approach :-

Text = " ABCAABCD" = n
 Pattern = " AAB" = m
 ↓
 j

```
int BruteForceStringMatch( char T[], char P[], int n, int m)
{
```

```
    for( int i=0; i<=n-m; i++)
    {
        int j=0;
        while( j<m && T[i+j]==P[j])
        {
            j++;
        }
        if( j==m)
            return i;
    }
}
```

$O(n \times m)$

$\begin{matrix} & i \\ 1 & 2 & 3 & 1 & 2 & 1 & 4 & 2 & 3 & 1 \\ & j \end{matrix}$	$n=10$ $m=4$
---	-----------------

$n-m$
 $10-4=6$

2) Rabin Karp :- $O(n+m)$

Hashfunction ($T(i, j), p$) $\rightarrow O(1)$ \rightarrow may be
 $(0, 4) \quad (1, 5)$ $m=4$ $\underline{\underline{}}$
 Not possible

$$T = \frac{1 \overset{1}{2} 3 \overset{1}{1} 2 \overset{1}{1} 3}{j}, \quad (121) \rightarrow p \quad m=3$$

$2 \times 10 + 3 = 23 \times 10 + 1$

$$1 \times 10 + 2 = 12 \times 10 + 3 = (123) \quad = 231$$

if $(T \% 10 == p \% 10)$

{ may be } $T = 123 \% 10 = 3$
 $P = 121 \% 10 = 1$

$123 \% 100$
 $23 \times 10 + 1$
 $\boxed{231}$

$$O(n+m)$$

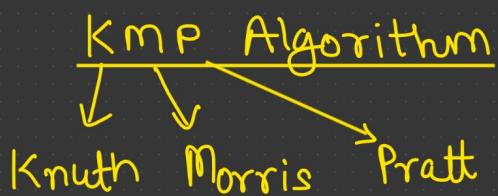
$T = 231 \% 10 = 1$
 $P = 121 \% 10 = 1$ \rightarrow may be

$$12 \% 10 = 2 \times 10 = 20 + 1 = 21$$

$231 \% 100 = 31 \times 10 + 2$
 $= \boxed{312}$

$$\underline{123 \% 100} = 23 \times 10 = 230 + 1 = 231$$

$$5641 \% 1000 = 641 \times 10 = 6410 + 1 = 6411 \rightarrow O(1)$$



Prefix Table

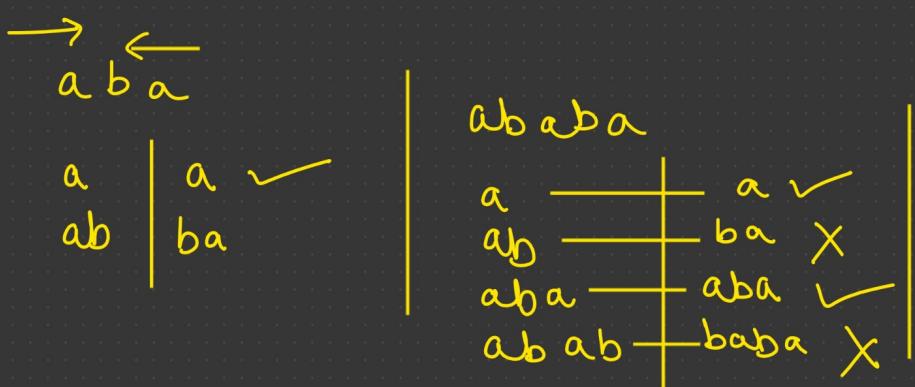
Ex:-

	j		i
o	1	2	3
P	a	b	a
F	0	1	2

```

if(j==0)
{
    F[i]=0
    i++
}
if(j>0)
    j=F[j-1]

```



```
Void prefixTable( int p[ ], int m )
```

int i=1, j=0, F[0]=0;

$\rightarrow O(n)$

while ($i < m$)

۹

if ($p[i] == p[j]$)

{

$$F[i] = j + 1;$$

j++ ;

j++;

乙

```
else if(j > 0)  
{
```

$$j = F[j-1];$$

۴

$\text{dx} \{ F[i] = 0;$
 $i++;$

3

```

int KMP( char T[], char P[], int n, int m)
{
    int i=0, j=0;
    prefixTable( P, m);  $\longrightarrow O(m)$ 

    while (i < n)  $\longrightarrow O(n)$ 
    {
        if (T[i]==P[j])
        {
            i++; j++;
            O(n+m)
            if (j==m)
                return i-j;
        }
        else if (j>0)
        {
            j = F[j-1];
        }
        else
            i++;
    }
    return -1;
}

```