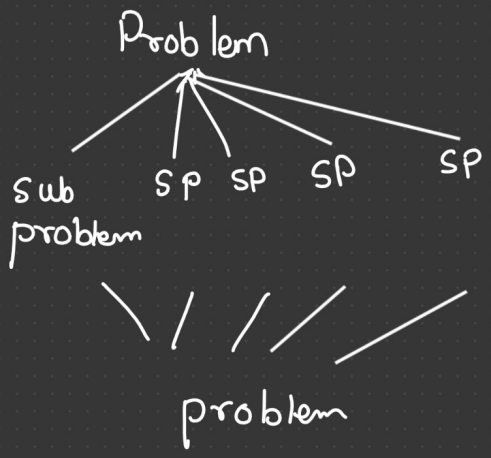
Dynamic Programming



Recursion

Fibonacci Series :-

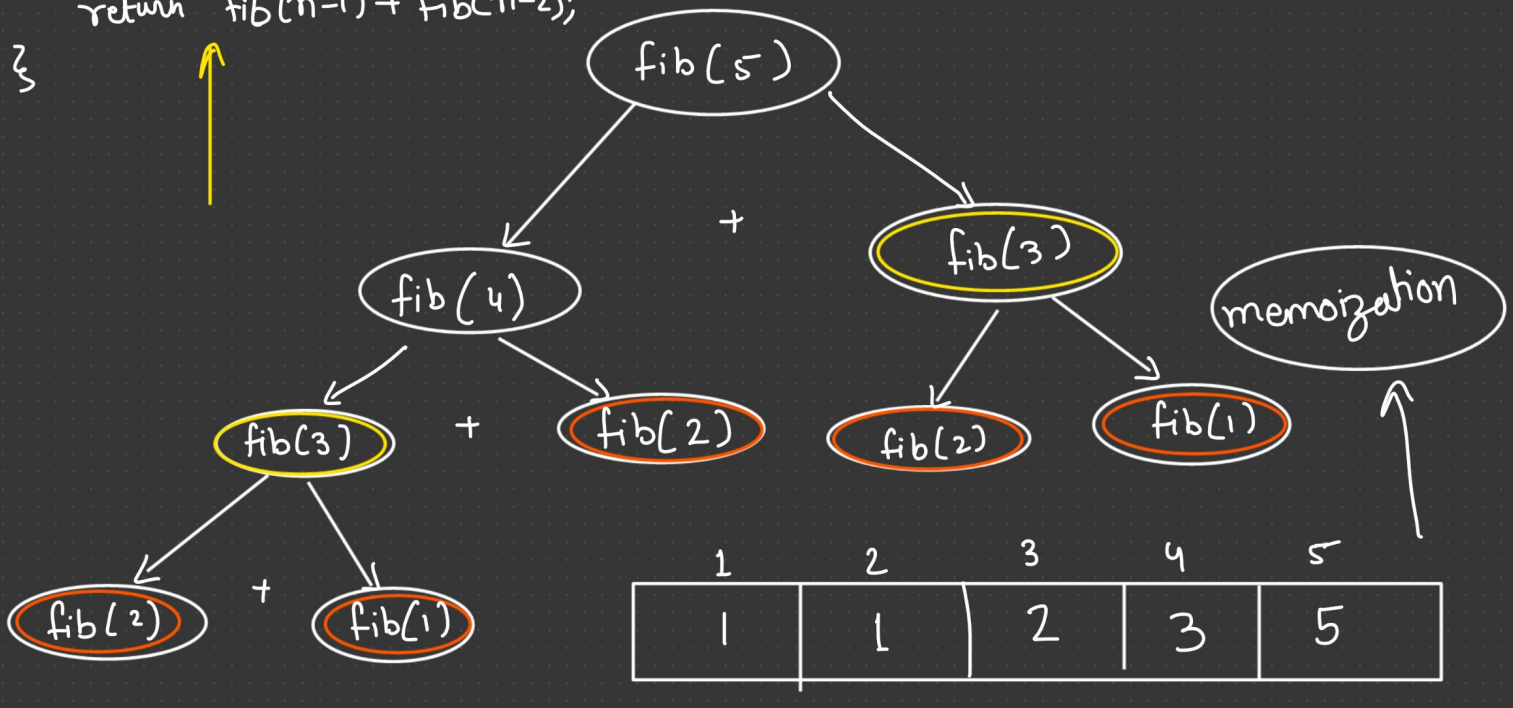
1, 1, 2, 3, 5, 8, 13, ...

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

$f(3) = fib(2) + fib(1) = 1 + 1 = 2$

$f(4)$ . . .

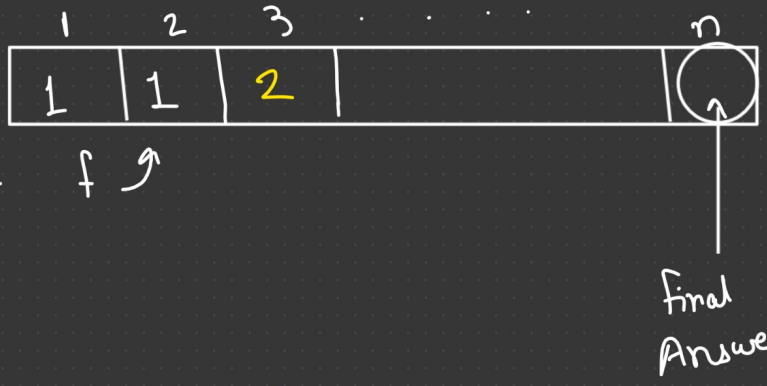
```
int fib ( int n)        (1)
{
  if (n == 1 || n == 2)
      return 1;
  else
  return fib(n-1) + fib(n-2);
}
```



```
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];
}
```

→ Top-down

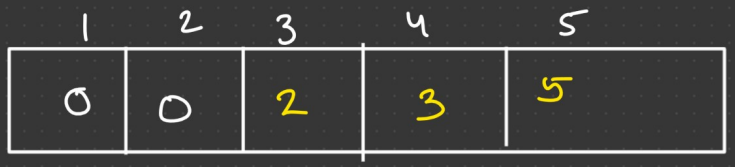
→ O(n)



```
int fib ( int n)
{
   if (n==1 || n==2)
      return 1;

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

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

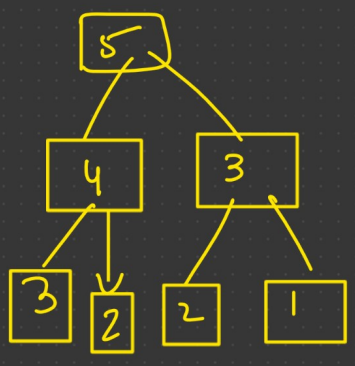
(2)

DP = Recursion + Memoization

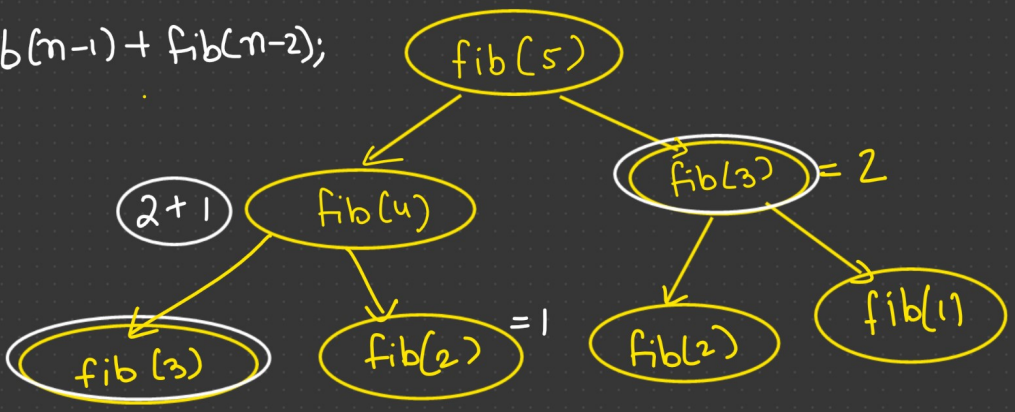


→ O(n)

→ O(n)

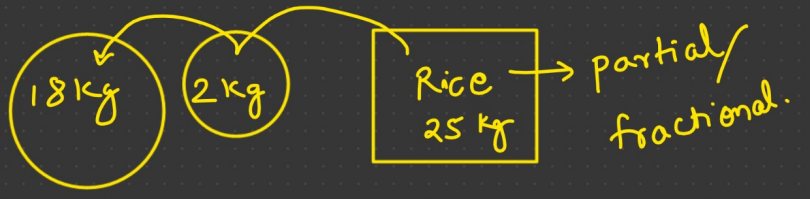


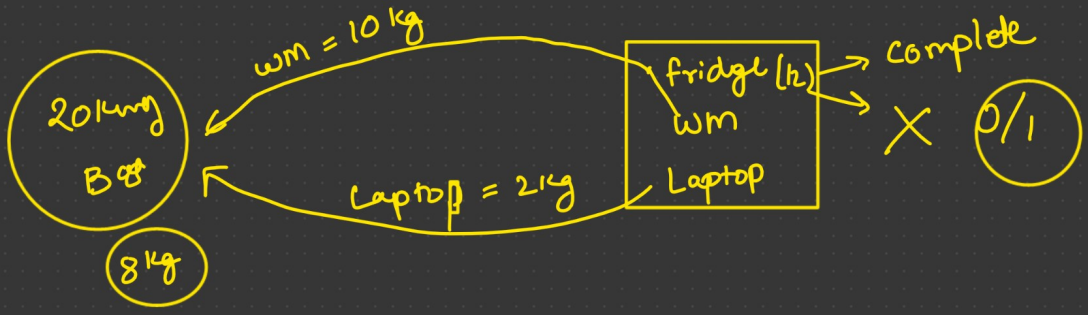
(2) =



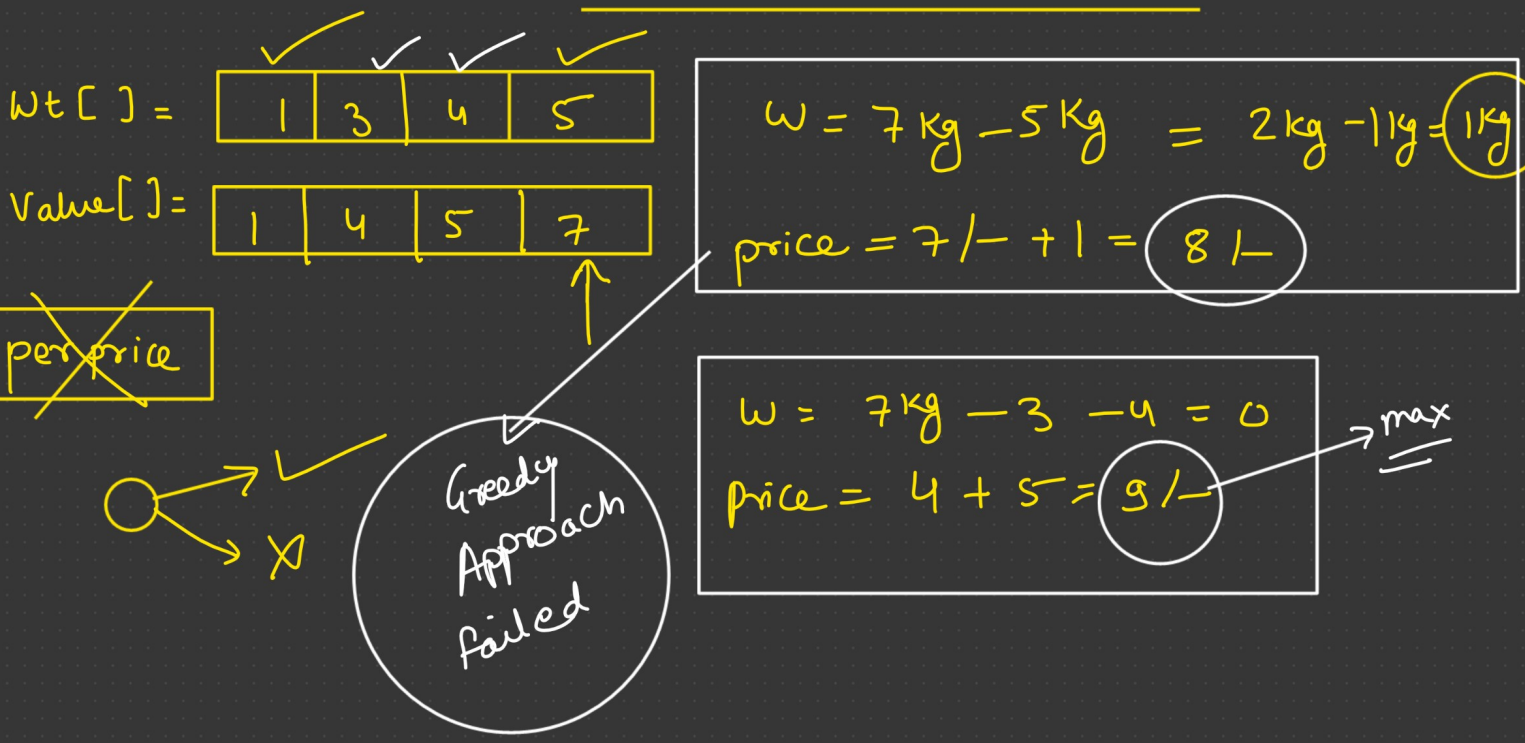
0/1 Knapsack problem

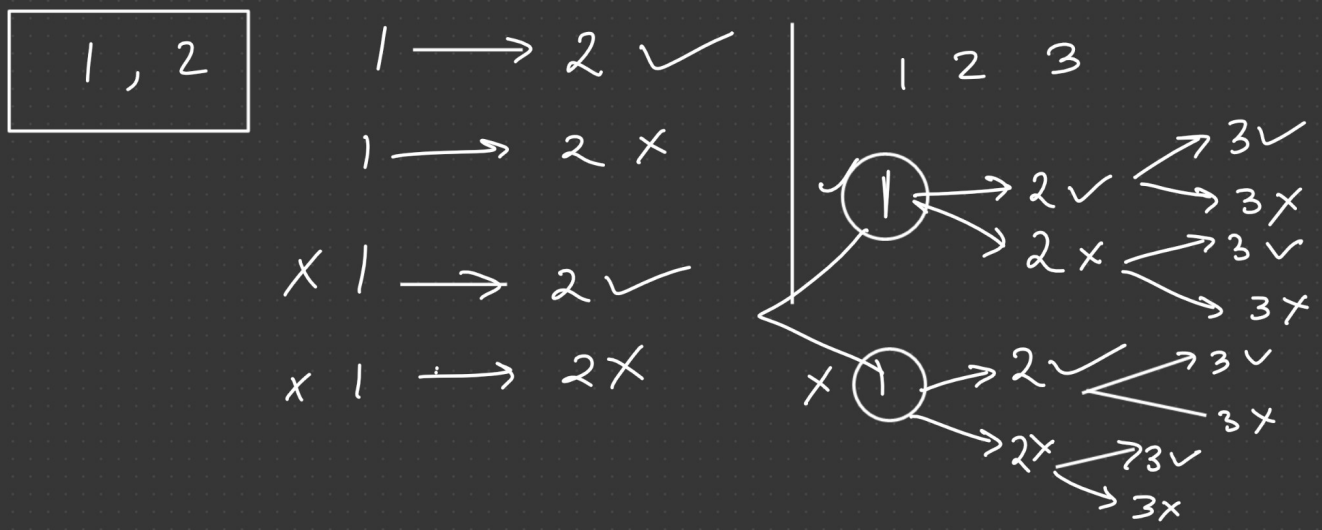
Knapsack problem
- 1) Fractional Knapsack Problem (Greedy Algorithm)
- 2) 0/1 Knapsack Problem → DP
- 3) Unbounded Knapsack problem.



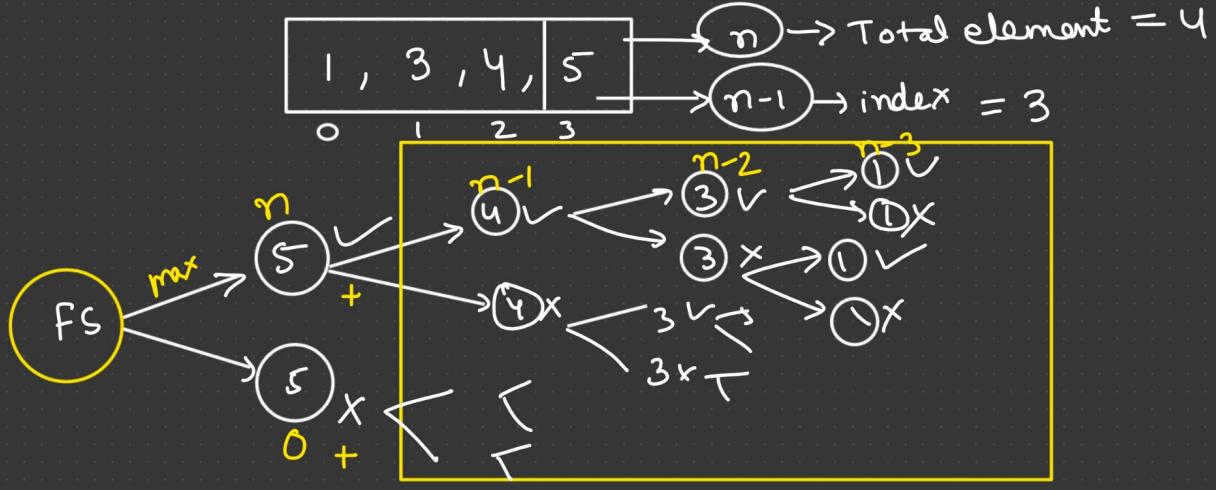


0/1 Knapsack Problem





It is growing exponentially



Given:-

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

Change

```
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);
  }
}
```

W = 0

n = 4 → product

3

	0	1	2	3
wt	1	3	4	5
value	1	4	5	7

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)
- wt (weight of n^{th} element)
- price (final price)

P	wt	0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1
4	3	2	0	1	1	4	8		
5	4	3	0						
7	5	4	0						

→ wt of Bag (w)

dp[2][5] (n, w)

→ price of Bag (max)

Ans = dp[n][w]

item no (n)

Top-Down approach

	0	1	2	3	4 → wt
0	0	0	0	0	0
1	0	4	4	4	4
2	0	4	4	7	7 ← i-1
3	0	4	4	7	(7) ← i

↓ n

1st

	0	1	2
wt	1	2	3
value	4	3	2

W = 4, max = 7/-

KS(4 , 3) (4 ← w, 3 ← n)

dp[i][w] = dp[i-1][w]

w++

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

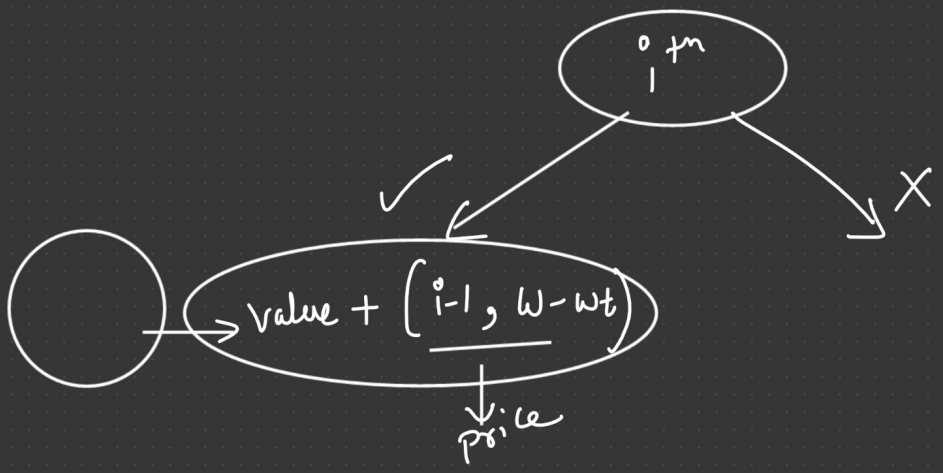
```
int knapsack ( int W, int n)
{
  for (int i=0 ; i<=n ; i++)
  {
    for (int w=0; w<=W ; w++)
    {
      if ( i==0 || w==0)
        dp[i][w] =0;
      else if ( wt[i-1]<= w)
      {
        dp[i][w] = max{ (value[i-1] + dp[i-1][w-wt[i-1]]) , dp[i-1][w] }
      }else
        d[i][w] = dp[i-1][w];
    }
  }
  return[n][w];
}
```

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

	0	0	0	0	0	0
0						
0						
0						

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

0th i → wt[i-1] value[i-1]



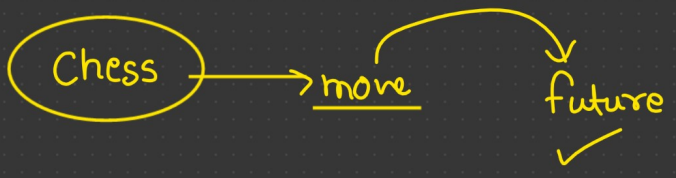
A	B	C	D
1	2	2	2
3	2	1	4

7

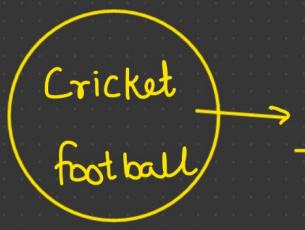
4 + [i-1, W-wt]

4 + 6 = 10

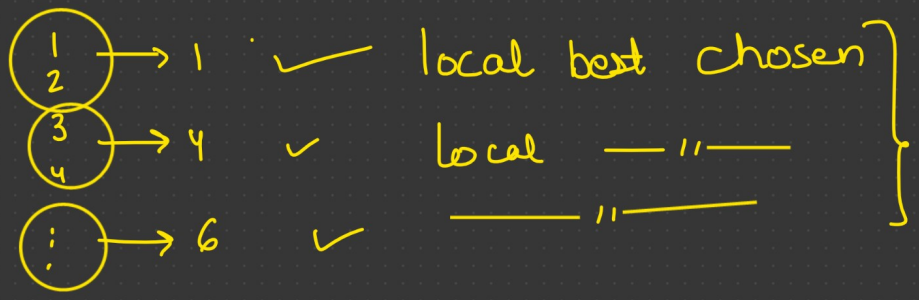
Greedy Algorithm



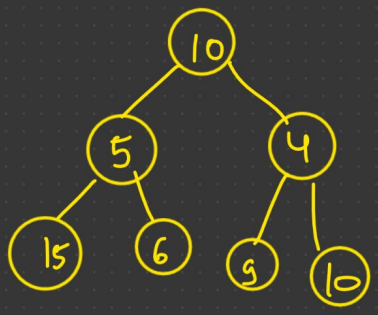
✗ This is not Greedy Approach



✓ This is an Greedy Approach.



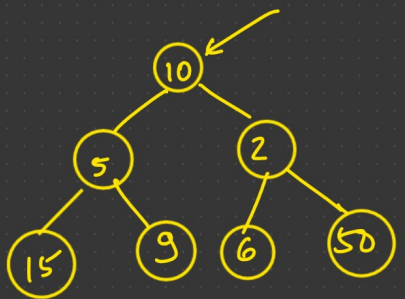
Problem :-



max sum ⇒ root → leaf

10 + 5 + 15 = 30

Advantage & Disadvantage :-



10 + 5 + 15 = 30

10 + 2 + 50 = 62 → ?

Greedy Approach failed

Ex :-

1) Sorting ⟶ Selection sort, Topological sort.

2) Priority Queue → Max Heap, min Heap

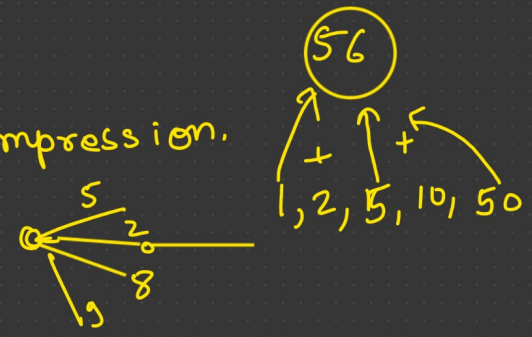
3) Huffman Encoding → Encryption, Compression.

4) Prim's, Kruskal's, Dijkstra's

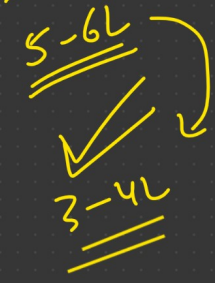
5) coin change problem

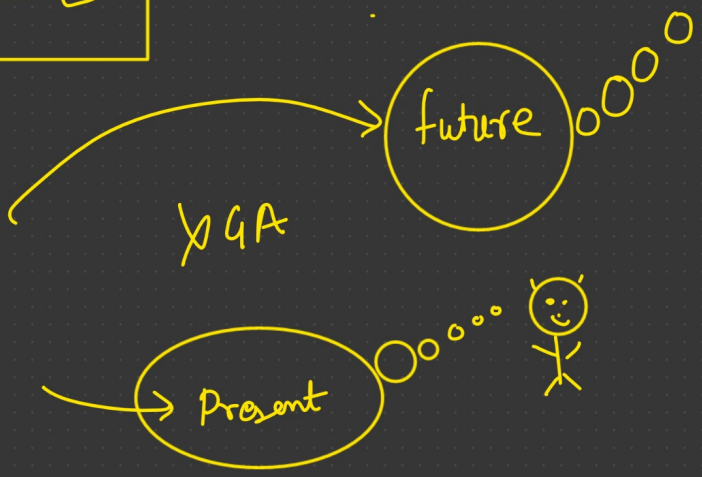
6) Fractional knapsack Problem.

7) Job scheduling algorithm.

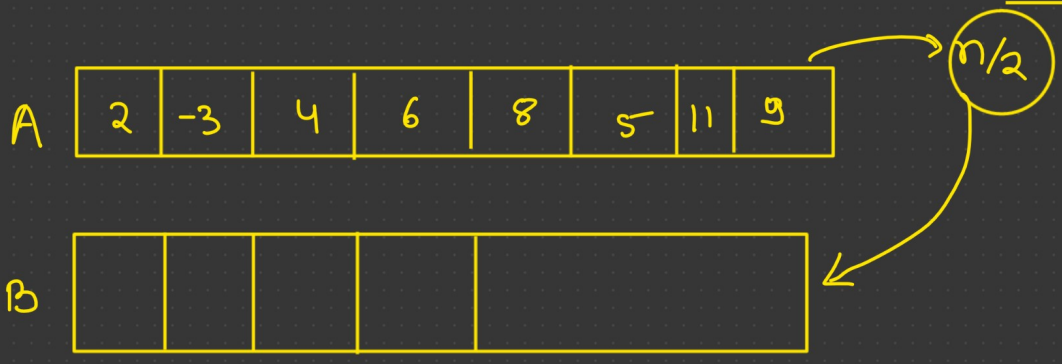


bag size limit





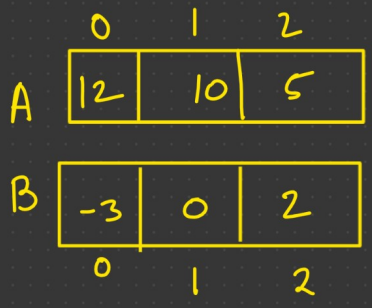
Problem: You are given an array A ,of n elements. You have to remove exactly n/2 elements from an array and add it to another array B (initially empty).Find the maximum and minimum values of difference between these two arrays. The difference between those two arrays is sum(abs(A[i]-B[i]).



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

Ex:-

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

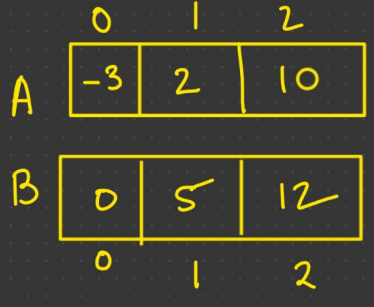


$$(A[0]-B[0]) + (A[1]-B[1]) + (A[2]-B[2])$$

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

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

$(12-(-3)) + (10-0) + (5-2) = 15+10+3 = 28$ → maximum



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

$3+3+2 = 8$ → minimum.

Maximum Sum = $(a_0 - b_0) + (a_1 - b_1) + (a_2 - b_2)$

$\downarrow$ max + $\downarrow$ max + $\downarrow$ max

A = 12, 10, 5, B = -3, 0, 2

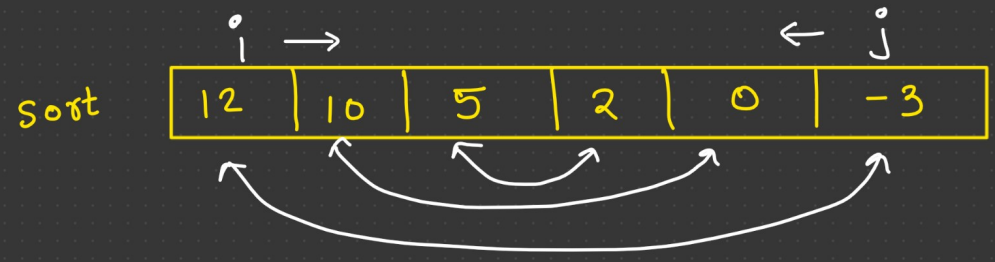
Step1:- Sort all the numbers in descending order.

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

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

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[2i] - a[2i+1]);
  j--;
}
cout << max << " " << min;
```

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

$3 - 0 = 3$

$3 - 1 = 2$

$3 - 2 = 1$

$3 - 3 = 0$ ← min

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

$2 + 3 + 3 = (8)$ ← 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];
```
} loop

```
int main()
{
    int deno[] = {1,2,5,10,20,50,100,500,2000};
    vector<int> ans; int n = 9; int Amt = 1583;
    sort(deno, deno+n);

    for(int i=n-1 ; i>=0 ; i--)
    {
        while( Amt >= deno[i])
        {
            Amt = Amt - deno[i];
            ans.push(deno[i]);
        }
        if( Amt == 0)
            break;
    }
        for(int i=0 ; i<ans.size() ;i++)
            cout<<ans[i]<<" ";
}
```

→ $O(n^2)$

→ $n - n^2$

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 >= finish time of current event. Then select that event. Otherwise move to next event.

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

$3 \not> 4$ $5 > 4$ $8 > 7$ $6 \not> 9$ $8 \not> 9$

✓ $n = 3$

C++

```
Struct event
{
    int start;
    int end;
};
```

start	6	5	1	8	3	8
finish	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 );
```

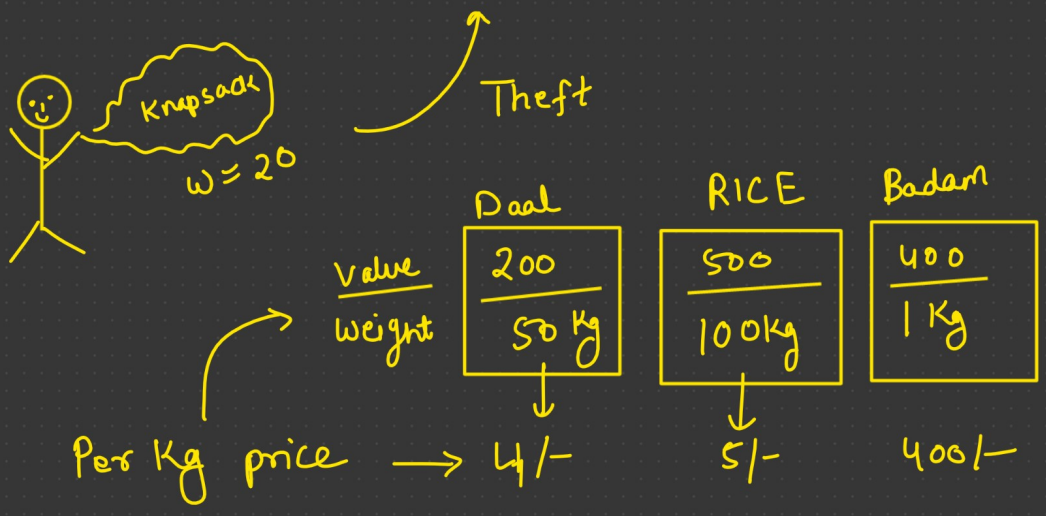
return type bool

```
  int current =0;
  cout << e[0].start <<"  "<<e[0].end;

   for [ int next =1 ; next < n ; next ++)
    {
       if ( e[current].end <= e[nex].start)
       {  coutt<< 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.

Value	21	24	12	40	30
Weight	7	4	6	5	6



Sort → Price (↑ ..... ↓)

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 decending 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

-5
15
-4
11
-6
5

price = 40 + 24 + 30 + 3(5) = 70 + 24 + 15 = 109/-

weigh = 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

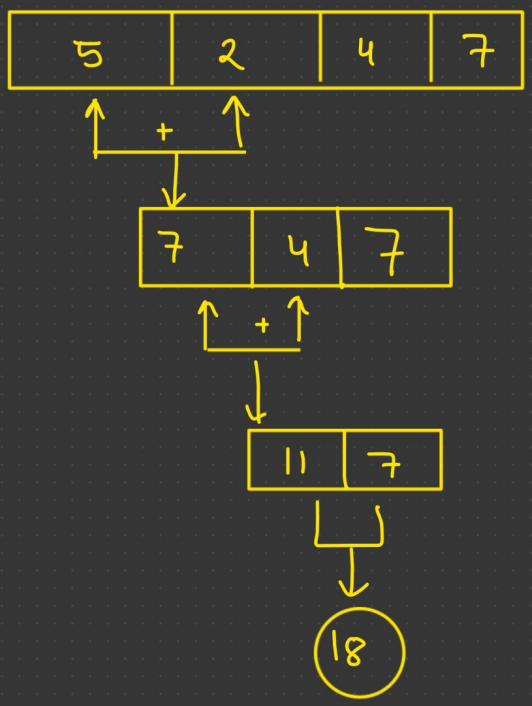
You are given n files, with their computation times in an Array. You can perform the following operation

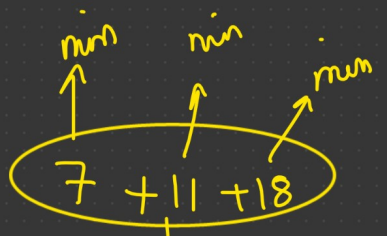
Operation:

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

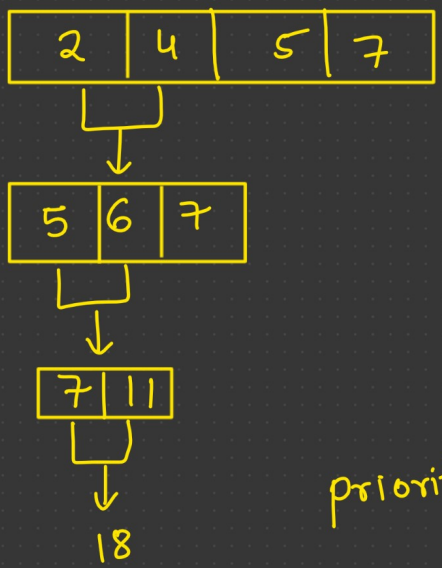
Do this until we are left with only one file in the array. We have to do this operation so that we can get the minimum cost finally.

ex:-





we need to minimize it



6+11+18

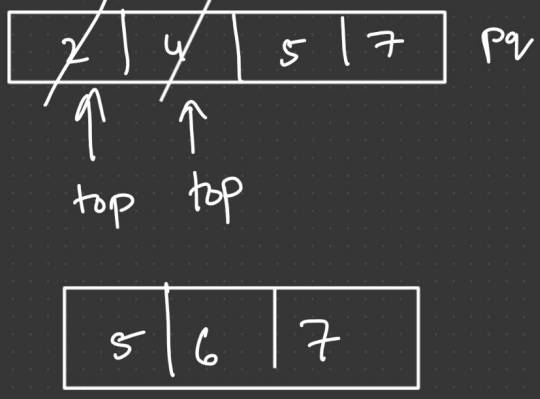
= 35

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;
}
```



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.

	T1	T2	T3	T4	T5	T6
arrival	9:00	9:40	9:50	11:00	15:00	18:00
departure	9:10	12:00	11:20	11:30	19:06	20:00

Railway station

3 train

```
if (pq.top < arr)
    pq.pop()
else
    count++
pq.push(dep);
```

No. of platform = 3

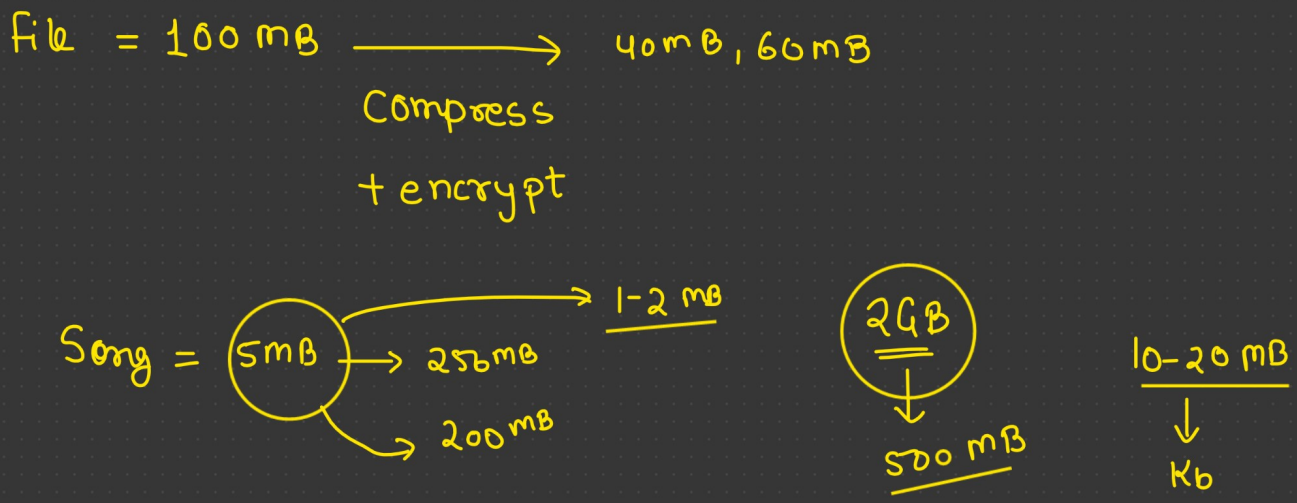
```
int main()
{
    int arr[] = {900, 940, 950, 1100, 1500, 1800};
    int dep[] = {910, 1200, 1120, 1130, 1900, 2000};

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

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

    sort ( V.begin(), V.end());

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

    int count =1;

    pq.push( V[0].second );

    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;
}
```

second.

first

→ insert departure time of 1st train

Efficient Huffman Coding for Sorted Input.



My name is prateek Jain. 1 char = 1 byte

= 25 = 25 bytes − 6 byte = 19 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

5 bit x3 = 15 bit

101

8-bit

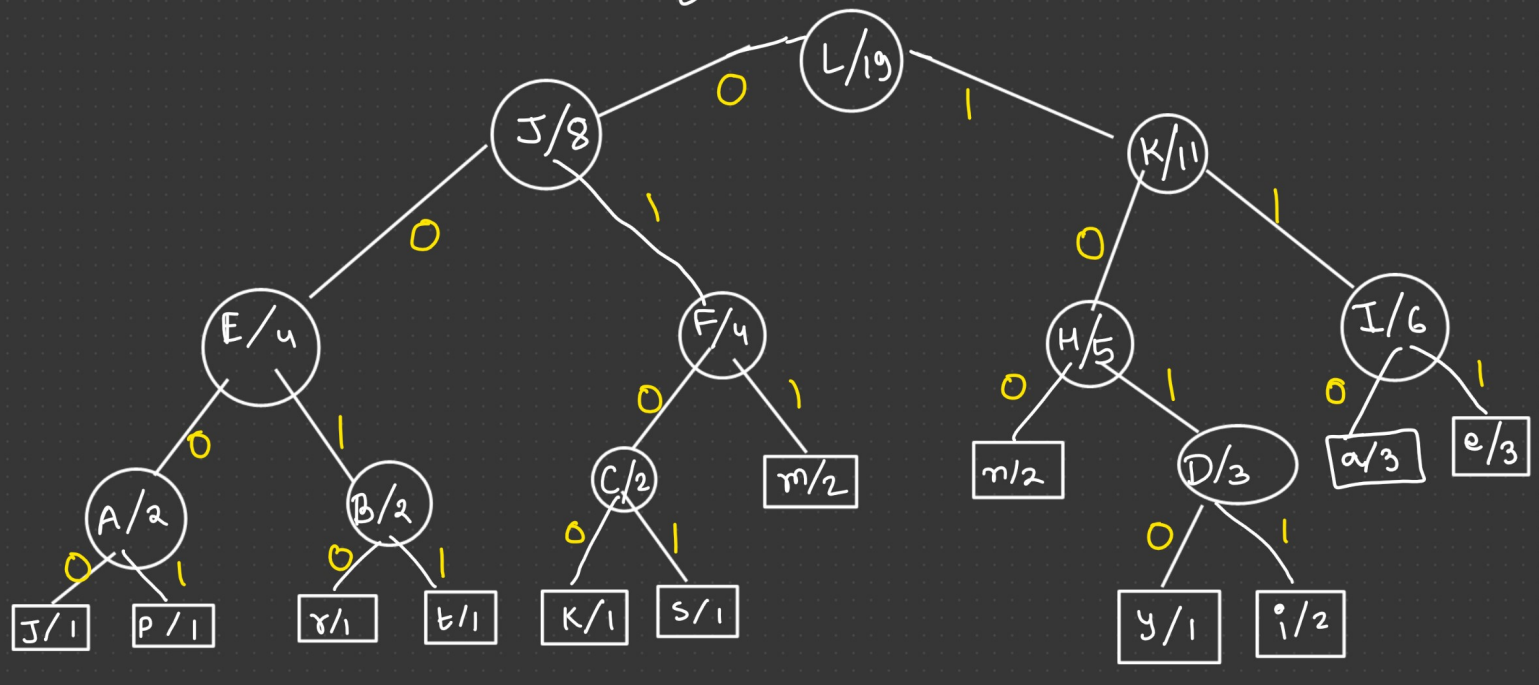
1 bit x 3 = 3-bit

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/2~~, ~~D/3~~, ~~E/4~~, ~~F/4~~, ~~H/5~~, ~~I/6~~, ~~J/8~~, ~~K/11~~



J = 0000	k = 0100	y = 1010
p = 0001	s = 0101	i = 1011
r = 0010	m = 011	a = 110
t = 0011	n = 100	e = 111

$= 7\times4 + 4\times3 + 2\times4 + 6\times3 = 28+12+8+18$

$= 40+26 = \frac{66}{8}$ bits

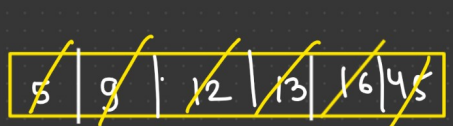
= 8 byte 2 bits

Implementing Huffman coding using 2 queue:-

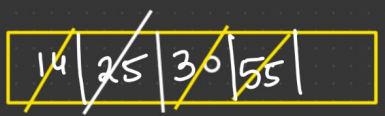
a	b	c	d	e	f
5	9	12	13	16	45

unsorted = $o(n\log n)$

sorted = $o(n)$

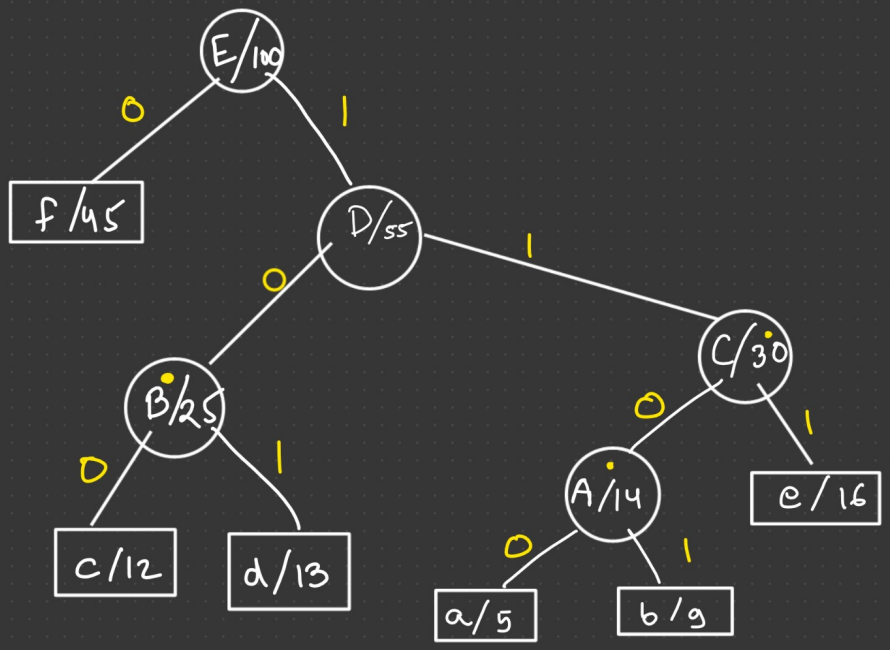


FQ



SQ

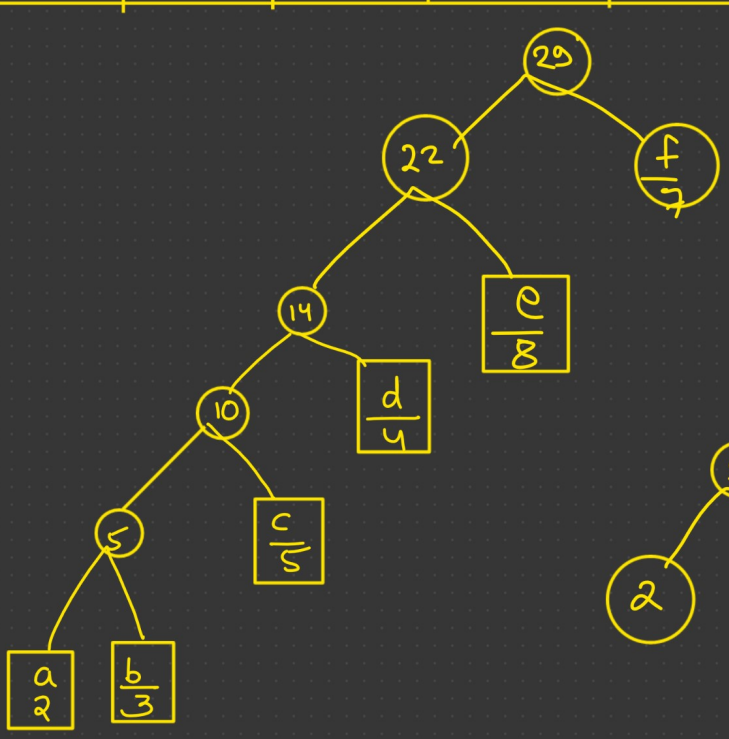
a - 1100
b - 1101
c - 100
d - 101
e - 111
f - 0



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.

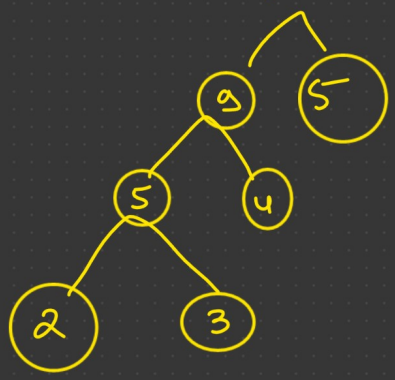
a	b	c	d	e	f
2	3	5	4	8	7

= 29



sort

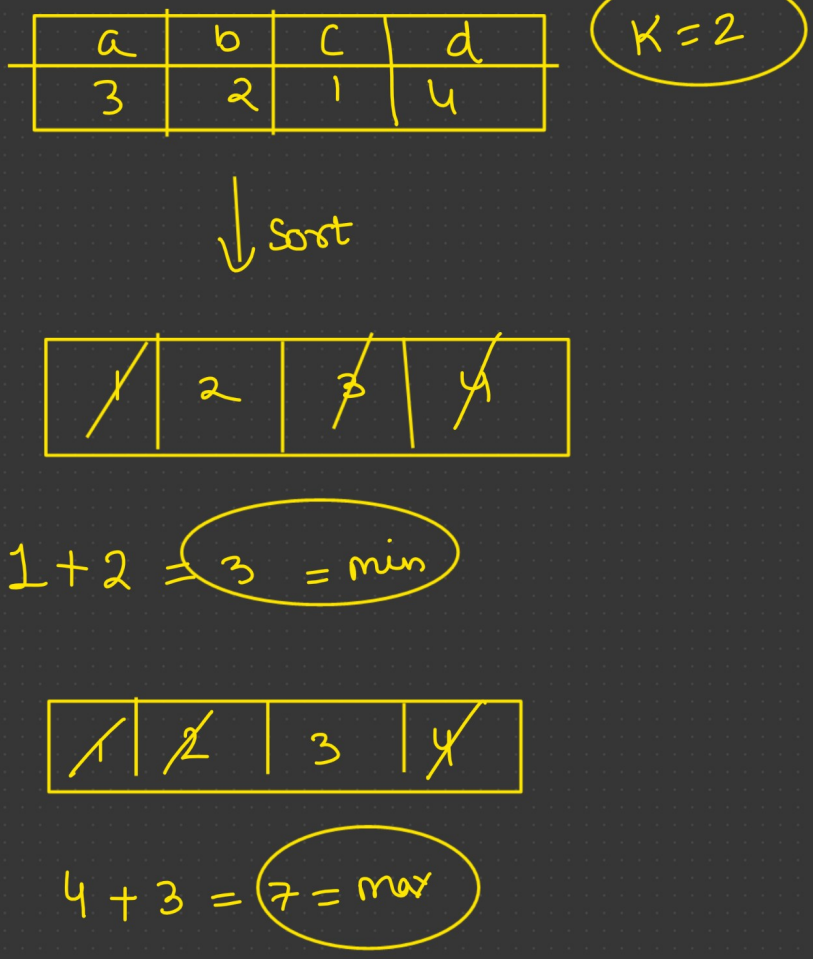
2	3	4	5	7	8



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

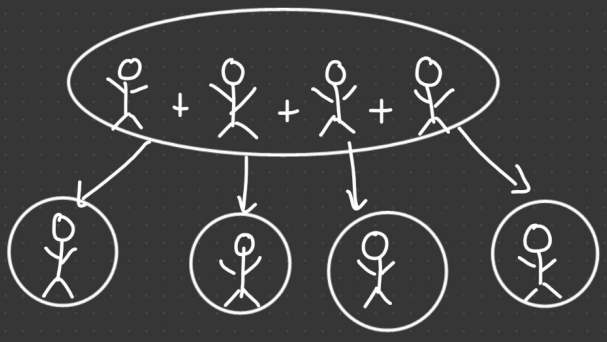



```
for(i=0; i<n; i++)
{
minprice = minprice + a[i];
    n = n-K;
}
```

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

Divide & Conquer

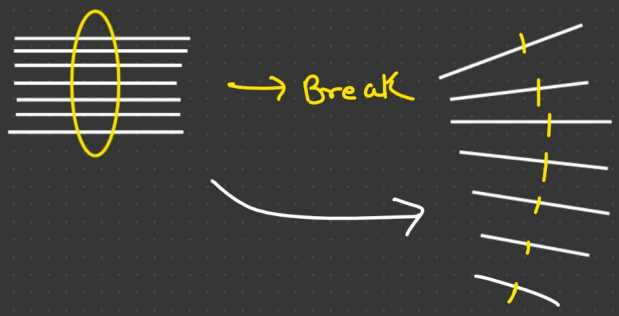
1) Binary Search
2) Merge Sort + Recursion
3) Quick Sort

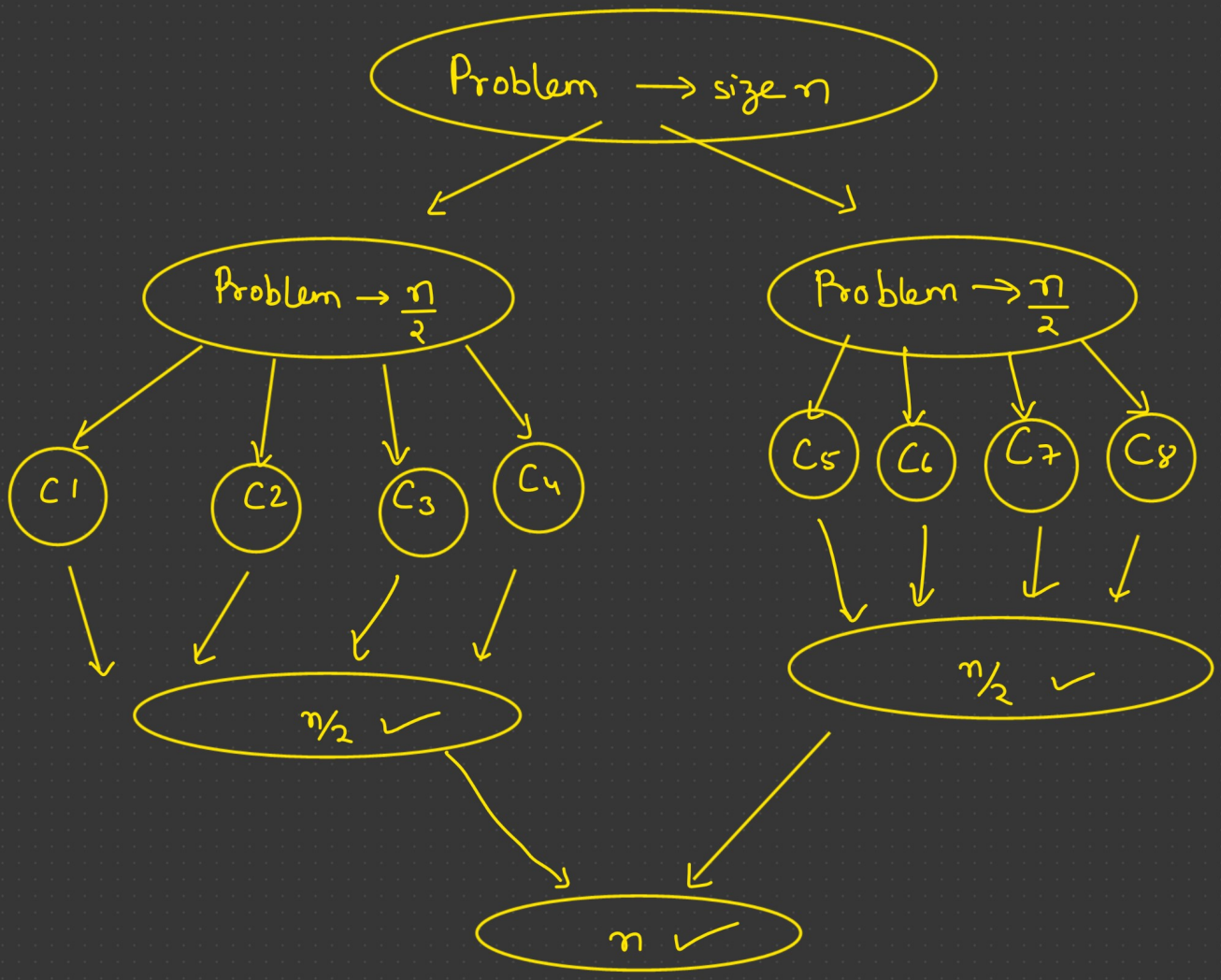


1 → stick
⋮
7 → stick

Father → 7 son's

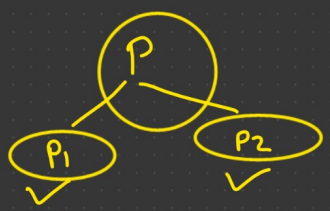
Property





Problem ⟶ D & C ✓

⟶ D & C ✗



Bigger Problem ⟶ small problem 1

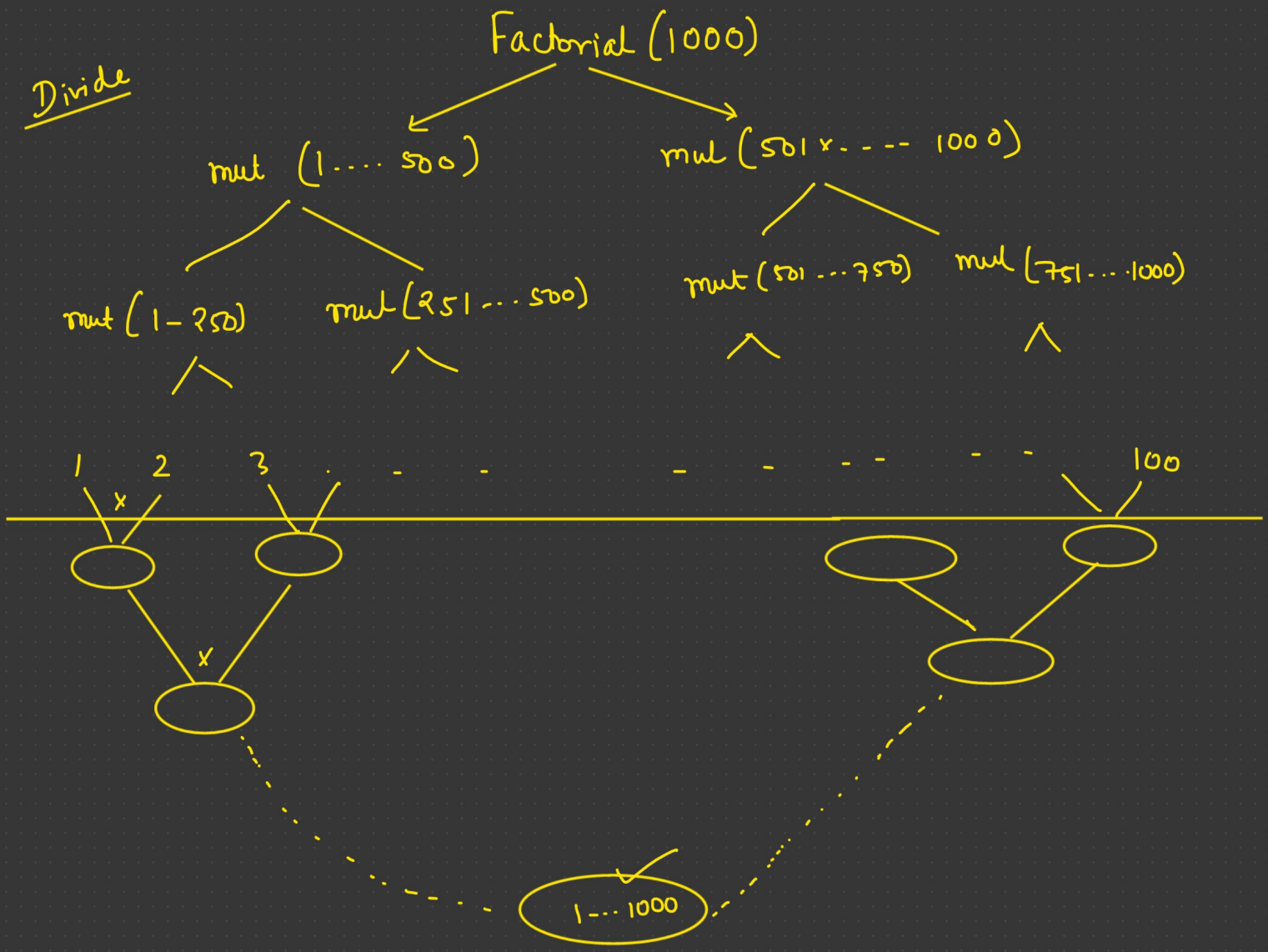
⟶ 2

⟶ 3

⟶ 4

* Each smaller problem must be independent of each other.

* Each smaller problem must be similar to the bigger problem.



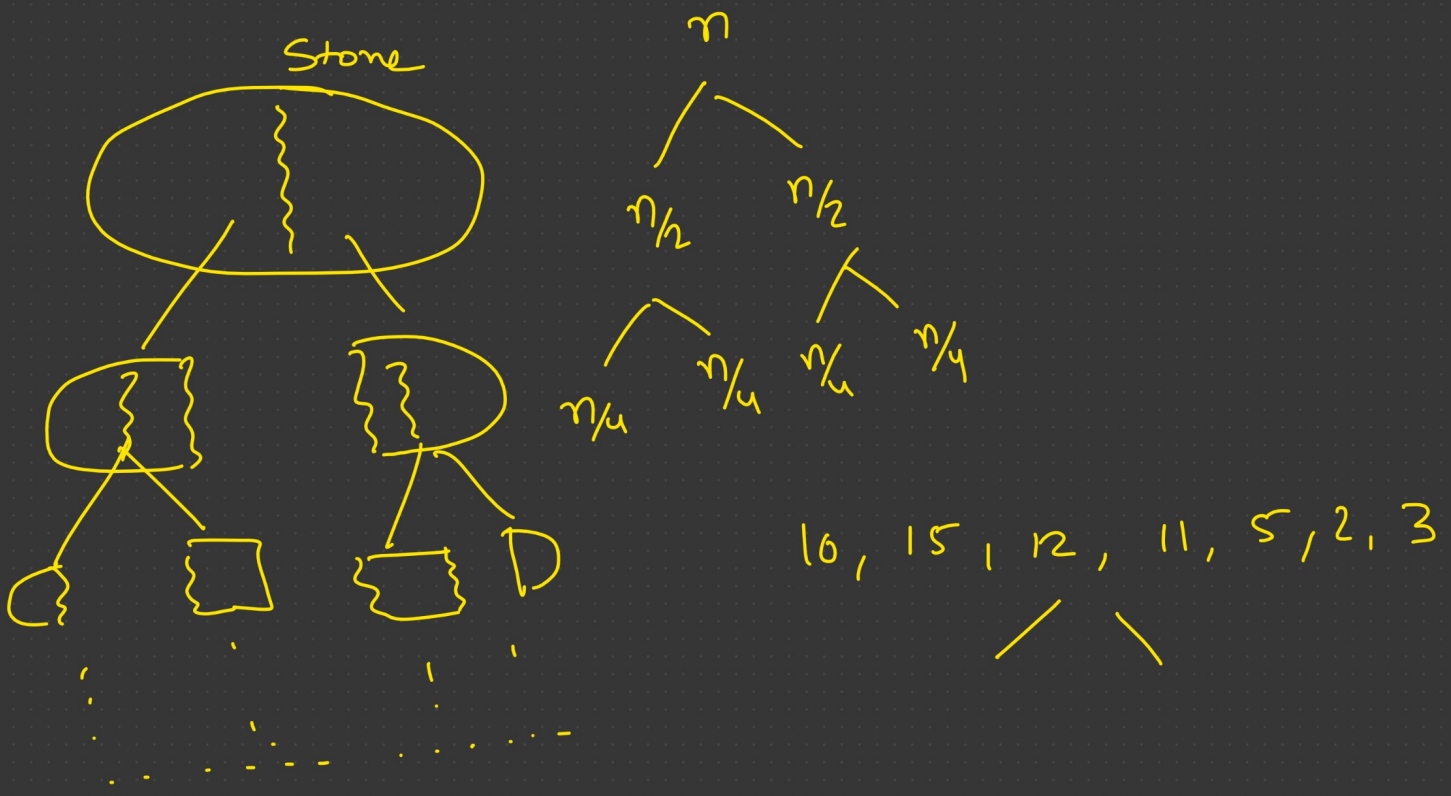
D&C Strategy ?

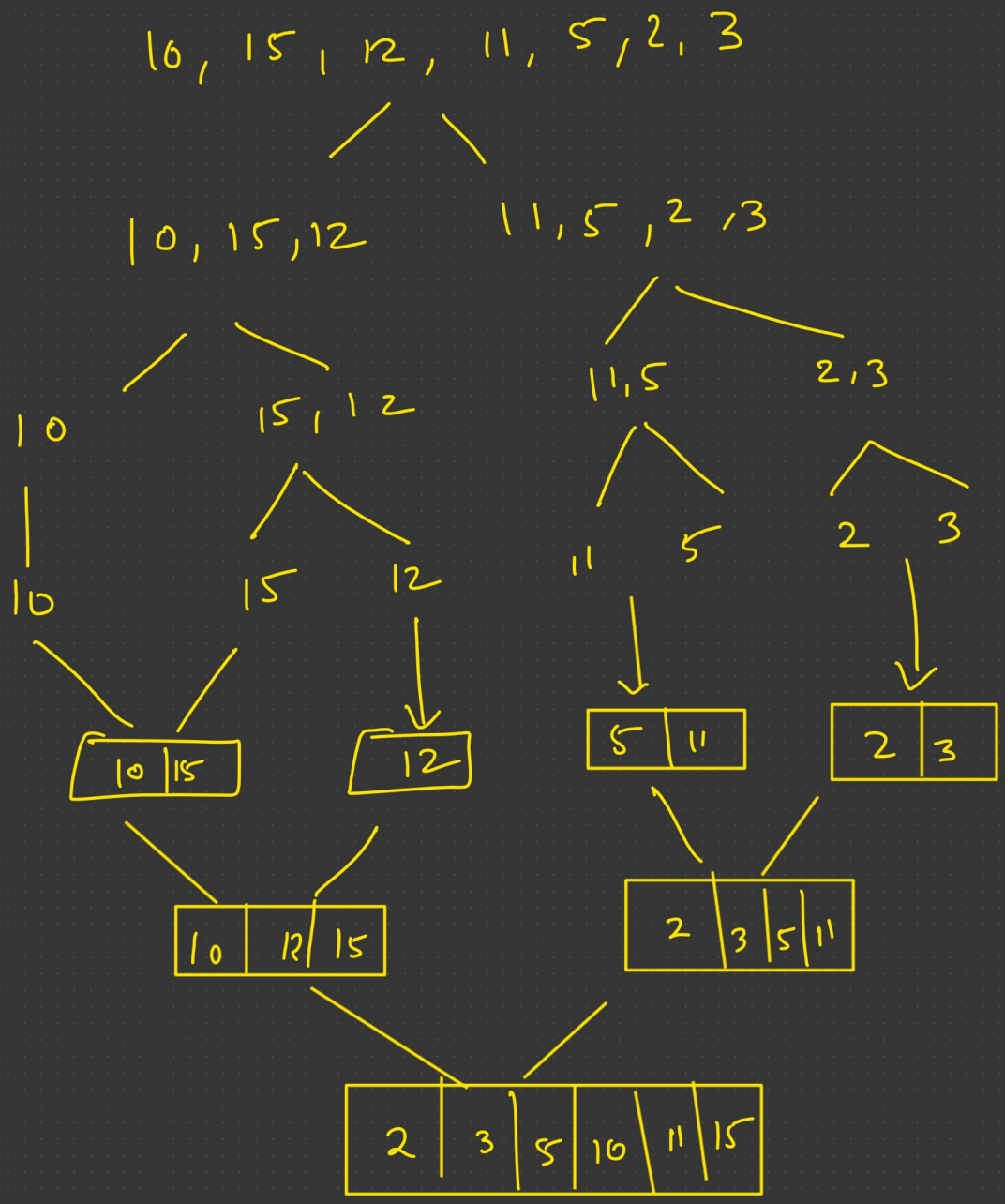
1) Divide — Breaking the problem into sub problems that are themselves smaller instance of the same type of problem.

2) Recursion — Recursively solving these sub-problems.

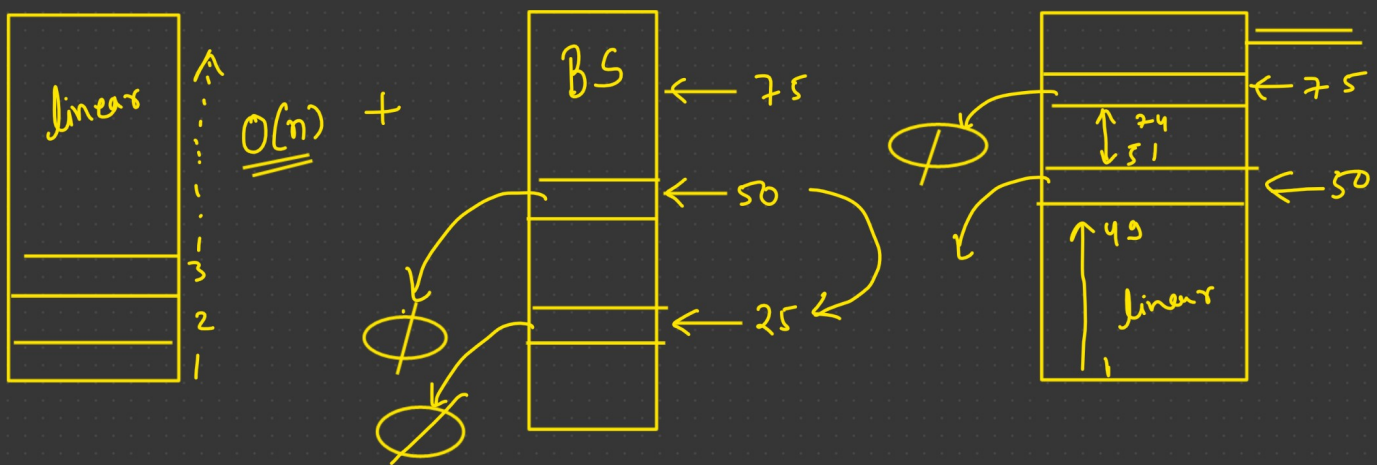
3) Conquer — Appropriately combining these results.

```
DandC(P)
{
    if(small(P))
      return Solution(P);

    // Divide the problem P into Subproblem(P1, P2, ....Pk)

    return Combine(DandC(P1),
                   DandC(P2),
                   DandC(P3),
                   :
                   DandC(Pk));
}
```

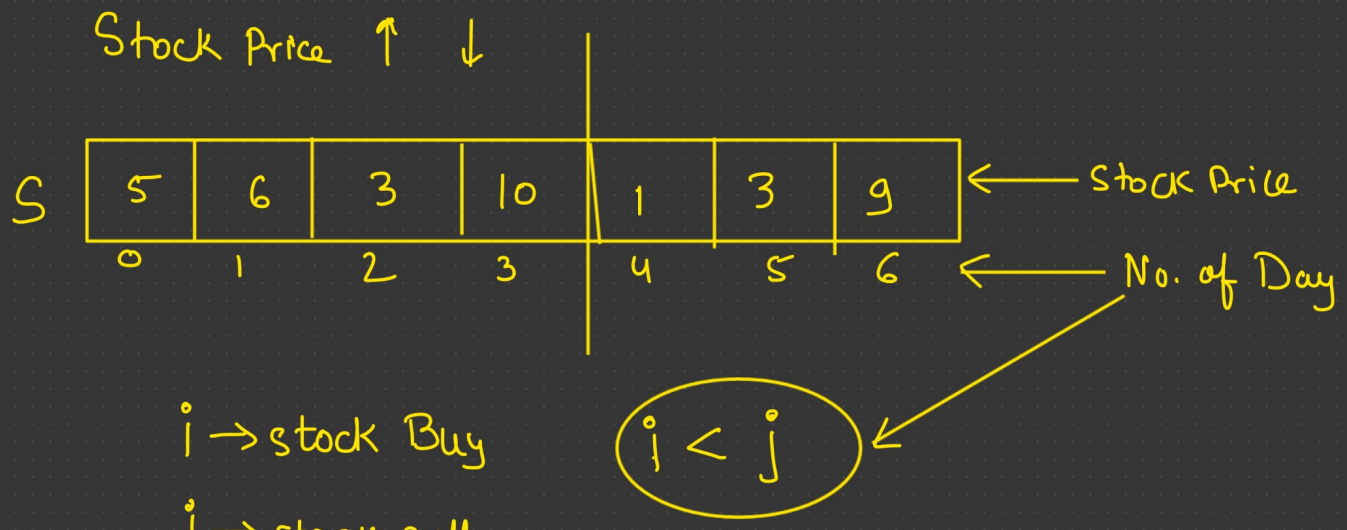




Ex:- You are given 2 eggs & 100 story building.



Maximize the profit for stock market :-



i → stock Buy

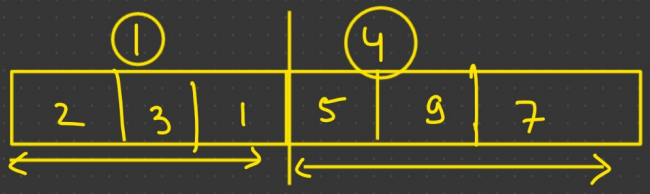
j → stock sell

Profit = S[j] - S[i] = S.P - C.P

1) Bruteforce :-

```
for (i=0; i<n-1; i++)
{
  for(j=i+1; j<n;j++)        ⟶ O(n²)
  {
    if ( Profit < S[j]-S[i])
        Profit = S[j]-S[i])
  }
}
```

2) Using D & C :-



9-1=8

a) left array MaxStock(S, left, mid) = (PL) = 1

b) Right array MaxStock(S, mid+1, right) = (PR) = 4

c) (PM) = max(Array[mid+1, right]) − min(Array[left, mid])

9 − 1 = 8

Profit = max(PM, PL, PR)

= max(8, 1, 4) = 8 → O(n log n)

DP → O(n)

Maximum SubArray Subsequence :-

i=2

−5, 6, −2, 11, −14, −5, 1

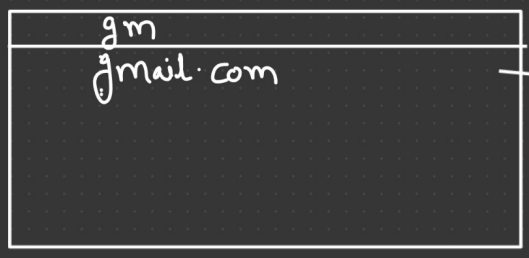
O(n log n) → D&C

max = ~~5~~ + ~~+6~~ 15

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

→ O(n²)

String Algorithms



→ Auto - completion.

Prateek

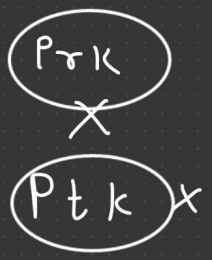
Prefix

P
Pr
Pra
Prat
Prate
Pratee
Prateek

Suffix

K
eK
eeK
teeK
ateeK
rateeK
PrateeK

Compare (prefix, Text list)
{
return list
}



Text = "My name is PrateeK" → string Matching

Pattern = "PrateeK"

1) Brute force approach →

2) Robin Karp

3) Kmp

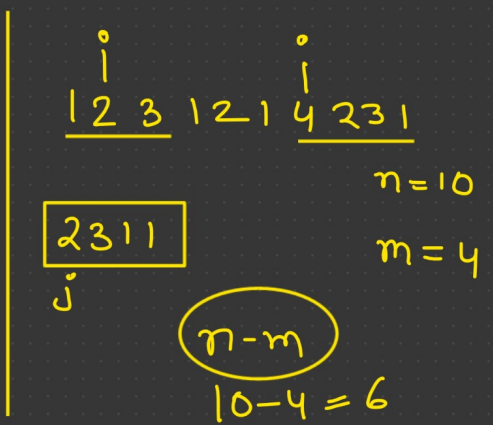
Brute force Approach :-

Text = "ABCAABCD" = n

Pattern = "AAB" = m

```
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;
    }
    return -1;
}
```

$O(n \times m)$



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

Hashfunction (T(i, j) , p) → O(1) → may be

- (0,4) (1,5)
- p → m = 4
- Not possible

T = 1 2 3 1 2 1 1 3 , 121 → p, m=3

i, j

1×10+2 = 12×10+3 = 123

2×10+3 = 23×10+1 = 231

if (T % 10 == P % 10) { } → may be → O(1)

T = 123 % 10 = 3

P = 121 % 10 = 1

123 % 100 = 23×10+1 = 231

$O(n+m)$

T = 231 % 10 = 1

P = 121 % 10 = 1 → may be

231 % 100 = 31×10+2 = 312

12 % 10 = 2 × 10 = 20 + 1 = 21

123 % 100 = 23 × 10 = 230 + 1 = 231

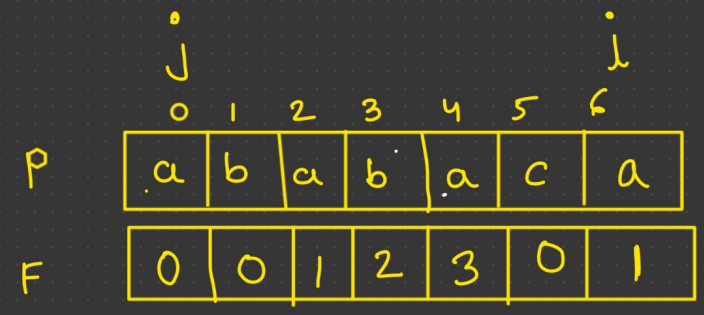
5641 % 1000 = 641×10 = 6410 + 1 = 6411 → O(1)

KMP Algorithm

Knuth Morris Pratt

Prefix Table

Ex:-




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

→ ← a b a

a	a ✓
ab	ba

a	a ✓
ab	ba ✗
aba	aba ✓
abab	baba ✗

```
Void prefixTable(int P[], int m)
{
    int  i=1, j=0, F[0]=0;

                                  ⟶ O(m)
    while(i<m)
    {
        if(P[i]==P[j])
        {
            F[i]=j+1;
            i++;
            j++;
        }
        else if(j>0)
        {
            j=F[j-1];
        }
        else { F[i]=0;
               i++;
        }
```

```
int KMP( char T[], Char P[], int n, int m)
{
    int i=0, j=0;
    prefixTable( P, m); ⟶ O(m)

    while (i<n)  ⟶ O(n)
    {
        if ( T[i]== P[i])
        {
            i++;  j++;

            if (j== m)
                return i-j;
        }
        else if (j>0)
        {
            j = F[j-1];
        }
        else
            i++;
    }
    return -1;
}
```

O(n+m)