Matrix Chain Multiplication

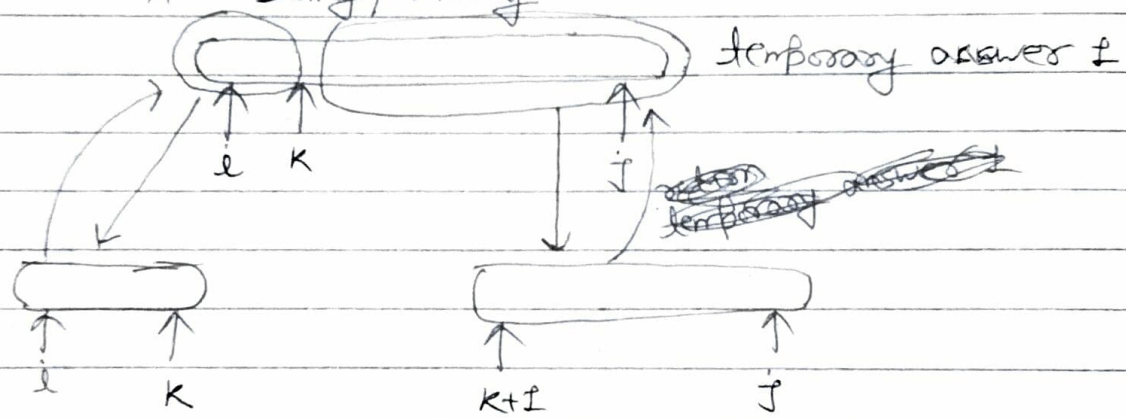
* Matrix Chain Multiplication -

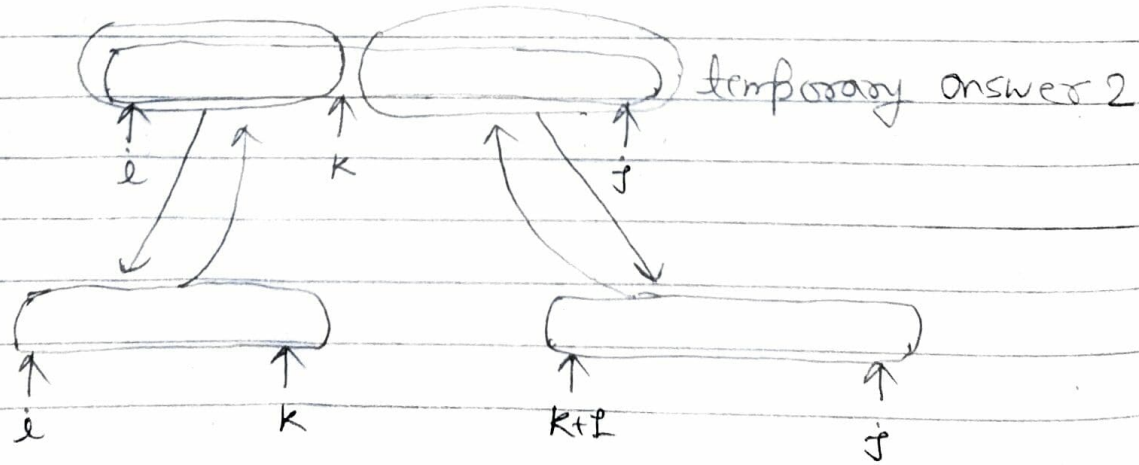
1- MCM → Recursive
→ Recursive with memoization

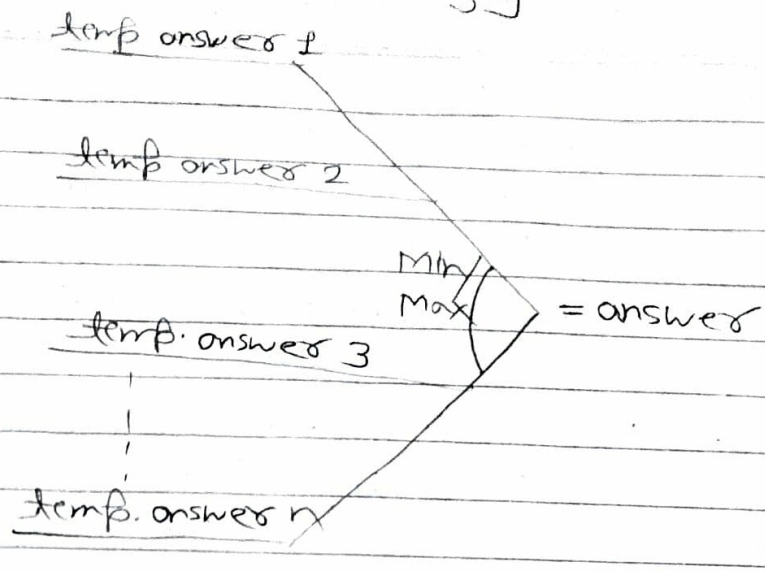
2- Print MCM
3- Evaluate expⁿ to True/Boolean parenthesis
4- Min/max value of an expⁿ
5- Palindrome Partitioning
6- Scrambled String
7- Egg Dropping problem

→ Identification + format -

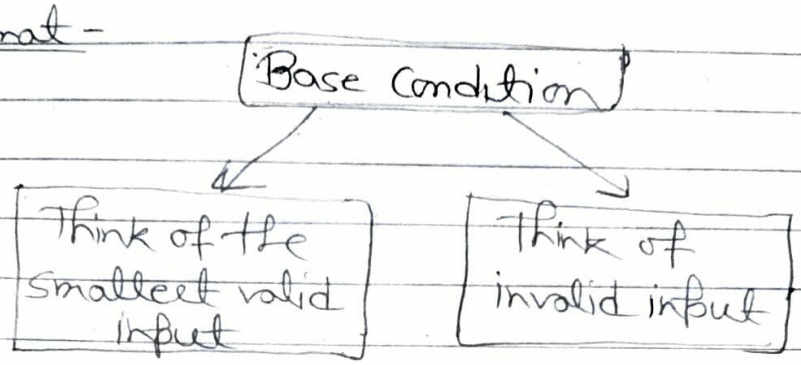
I/P : String / Array







Format -




```
int solve (int arr[], int i, int j)
{
    if (i>j)
      return 0
    for (int k=i ; k<j ; k++)
    {  // calculate temporary answer
       tempans = solve(arr, i, k)
                 solve (arr, k+1, j)
       ans = funct^n (tempans)
    }
    return ans
}
```

10,345,00

150
60
000
0

I- MCM × Recursive -

I/P $\boxed{arr[\,] = \{40, 20, 30, 10, 30\}}$ $\boxed{O/P: 26000}$

Given matrices $A_1(2\times5)$, $A_2(20\times20)$, $A_3(30\times20)$ - - - -
we have to multiply these matrices so that total cost will be minimum.
↓
No of multiplication

Cost - $[\quad]_{2\times3} \times [\quad]_{3\times6} = [\quad]_{2\times6}$

cost - 36 $(2\times3\times6)$

Let, $A \rightarrow 10\times30$
$B \rightarrow 30\times5$
$C \rightarrow 5\times60$

$A.(BC)$
$(10\times30)\cdot(30\times60)$

BC

$Cost(A(BC)) = (30\times5\times60) + (10\times30\times60)$
$= 9000 + 18000$
$= 27000$

$Cost((AB)C) = (10\times30\times5) + (10\times5\times60)$
$= 1500 + 3000$
$= 4500$ (min. cost)

→ arr[]: 40, 20, 30, 10, 30

A_1 = 40, 20 ; A_2 = 20, 30 ; A_3 = 30, 10 ; A_4 = 10, 30

For 5 elements there are 4 matrices
So, for n elements there are (n-1) matrices

Matrices are –

$A_1 = 40 \times 20$
$A_2 = 20 \times 30$
$A_3 = 30 \times 10$
$A_4 = 10 \times 30$

$A_i = arr[i-1] \times arr[i]$

→ Approach –

0	1	2	3	4
40	20	30	10	30
	↑ i			↑ j

```
int solve (int arr[], int i, int j)
{ if (i>=j)
      return 0;
  int mn = INT_MAX;
  for (int k=i ; k<=j-1 ; k++)
  {
  int tempans = solve (arr, i, k) +
                solve (arr, k+1, j) + arr[i-1]
                * arr[k] * arr[j];
    if (tempans < mn)
        mn = tempans;
  }
  return mn;
}

solve (arr, 1, n-1)
```

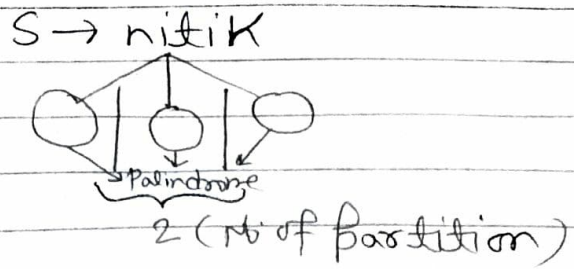
Cost to multiply (A_1A_2) to (A_3A_4)

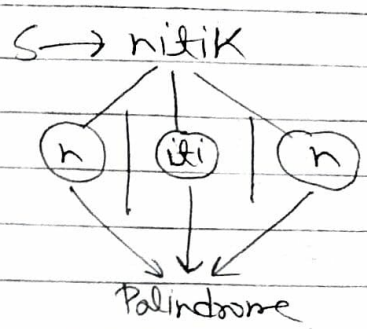
MCM memoization

```
int static t[1001][1001];
int solve (int arr[], int i, int j)
{ if (i>=j)
      return 0;
  if (t[i][j]!=-1)
      return t[i][j];
  int mn = INT_MAX;
  for (int k=i; k<j; k++)
  {
  int tempans = solve(arr, i, k) + solve(arr,
                k+1, j) + arr[i-1]* arr[k] *
                arr[j];
  if (tempans<mn)
     mn = tempans;
  }
  return t[i][j]=mn;
}
memset (t, -1, sizeof(t));
solve (arr, 1, n-1)
```

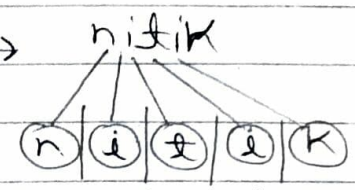
5- Palindrome Partitioning Recursive -

S: nitiK O/P: 3





In worst case →



No of partition = 4 (n-1)

Base condition-

```
if( i>=j )
    return 0;
if( ispalindrome (s, i, j))
    return 0;
```

Code-

```
int solve (string s, int i, int j)
{ if(i>=j)
    return 0;
  if (ispalindrome(s, i, j))
    return 0;
  int mn = INT_MAX;
```

```
for (int k=i ; k<= j-1 ; k++)
{
  int temp = 1+solve(s, i,k) + solve(s,k+1, j);
  if (temp < mn)
     mn = temp;
}
return mn;
}
```

Palindrome Partitioning x memoization -

j →

i ↓

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

t[i][j]

```
int static t[1000][1000];
int solve (String s, int i, int j)
{  if (i>=j)
      return 0;
   if(isPalindrome (s, i, j) == True)
      return 0;
   if(t[i][j] != -1)
      return t[i][j];
   int mn = INT_MAX;
   for (int k=i; k<= j-1 ; k++)
{
   int temp = 1+solve (s, i, k) + solve(s, k+1, j);
   if (temp < mn)
      mn = temp;
}
```

```
return t[i][j] = mn;
}
```

Optimization Palindrome Paritioning -

```
int static t[1001][1001];
```

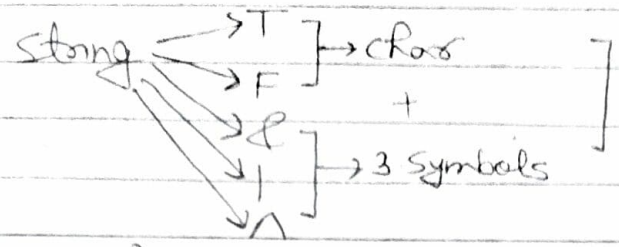
```
{ int temp = 1 + solve(s, i, k) + solve(s, k+1, j); }
```

⇓

```
if (t[i][k] != -1)
    left = t[i][k];
else {
    left = solve(s, i, k); t[i][k] = left; }
if (t[k+1][j] != -1)
    right = t[k+1][j];
else {
    right = solve(s, k+1, j);
    t[k+1][j] = right;
    }
int temp = 1 + left + right;
```

* 3. Evaluate Expression to True Boolean Parenthesization Recursive

I/P [String : T xor F and T] O/P [2]



we have to manage brackets b/w characters of string so that output must be 'True'.

Output → No of ways we can arrange brackets, so that output must be true.

T xor F and T

((T ∧ F) & T)
T & T
= True

(T ∧ (F & T))
T ∧ F
= True

Step 1 Find i & j

$i = 0$

$j = s.length - 1$

Step 2 Base condition

Exp 1 $\boxed{\text{xor}}$ Exp 2

No of True ways = left T * right F + left F * right T

Exp 1 xor Exp 2

No of ways this is True ⟷ No of ways this is False

or

No of ways this is False ⟷ No of ways this is True

overall True

```
if (i > j)
    return False;
if (i == j){
    if (isTrue == True)
        return s[i] == True;
    else
        return s[i] == False;
}
```

isTrue()
returns True if we want to calculate for True
else return False.

Step 3- Find k Loop

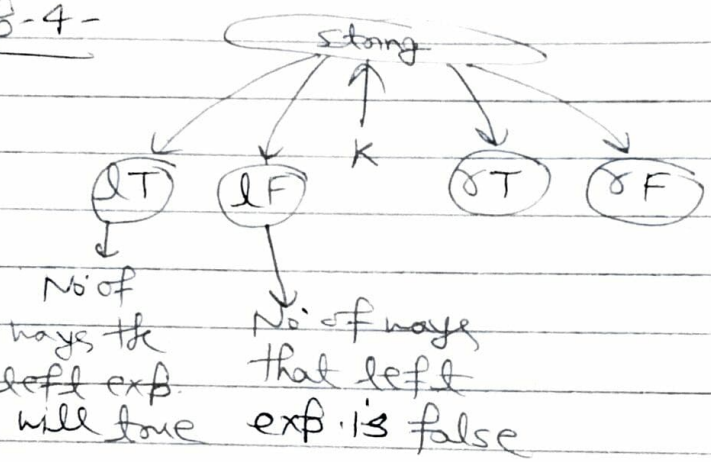
T or F and T xor F

k k k

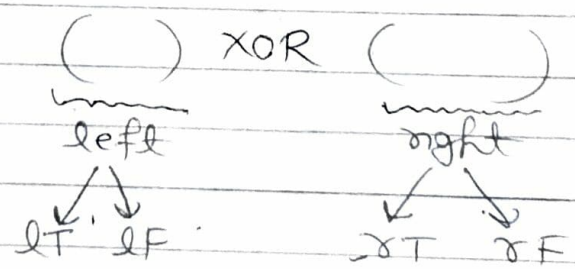
```
for (int k = i+1; k<= j-1; k+=2)
{
int lT = solve(s, i, k-1, T)
int lF = solve(s, i, k-1, F)
int rT = solve(s, k+1, j, T)
int rF = solve(s, k+1, j, F)
```

Temporary answer

Step-4-



For XOR



(lT * rF + lF * rT)

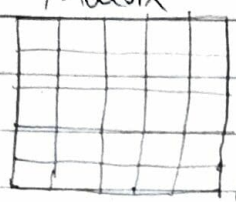
```
if( s[k] == '&')
{ if ( isTrue == True)
      ans = ans + lT * rT;
  else
      ans = ans + lF * rT + lF * rF + lF * rF;
}
elif ( s[k] == '|')
{
  if(isTrue == True)
      ans = ans + lT * rT + lT * rF + lF * rT;
  else
      ans = ans + lF * rF;
}
elif (s[k] == '^')
{
  if(isTrue == True)
      ans = ans + lF * rT + lT * rF;
  else
      ans = ans + lT * rT + lF * rF;
}

return ans;
```

Evaluate expression to True Boolean Parenthesization Memoization:-

I/P ~~ster~~	O/P
String: T xor F and T	2

Here i, j & isTrue are changing (acc. to previous question)

Matrix



(i x j x isTrue)
(3D Matrix)

Map

key	value
30 25 F	2
"	"

i+" "+j+" "+isTrue

```
int t[1001][1001][2];
unordered_map<string,int>mp;
```

```
string temp = to_string(i);
temp.push_back(" ");
temp.append(to_string(j));
temp.push_back(" ");
temp.append(to_string(isTrue));
if (mp.find(temp) != mp.end())
    return mp[temp];
```

→ after base condi^n but this code

```
mp[temp] = ans;
```

→ before return ans, put this code.

* 6- Scrambled String (Recursive)

I/P
a: "great"
b: "rgeat"

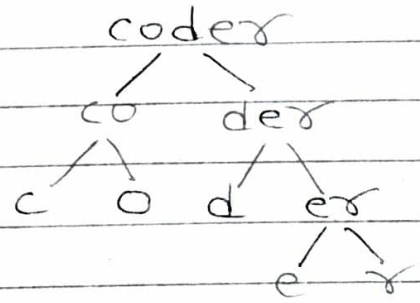
O/P
True

Given two strings s1 & s2 of equal length, the task is to determine if s2 is a scrambled form of s1.

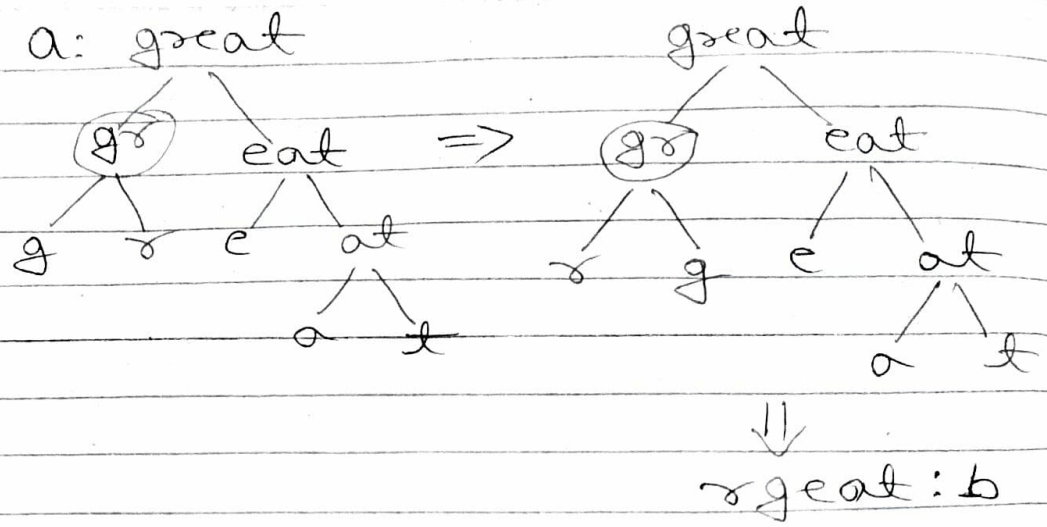
Scrambled String -

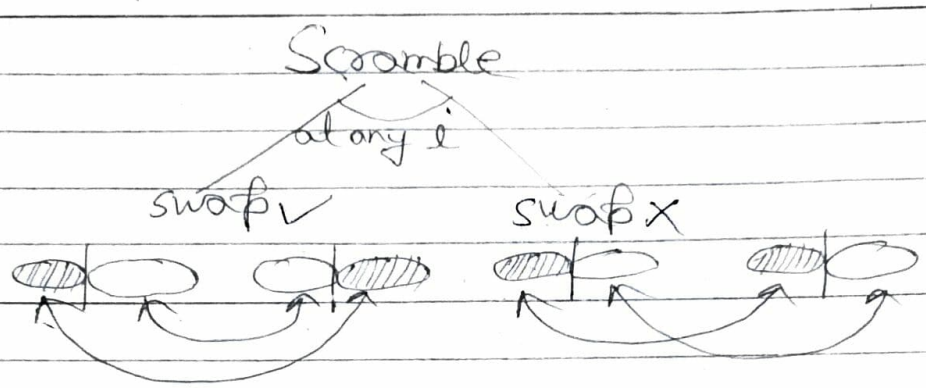
Given string str, we can represent it as a binary tree by partitioning it into two non-empty substrings recursively.

Let s = "coder"



To scramble the string, we may chose any non-leaf node & swap its two children.





<u>Case-I-</u> (swap ✓)

great eatgr bool solve(a, b)
0 1 2 3 4 0 1 2 3 4

if (solve (a.substr (0, i), b.substr (n-i, i)))
length length

True (scrambled)

&&

solve(a.substr (i, n-i), b.substr (0, n-i))

True (scrambled)

Then string (great & eatgr are scrambled)

Case-II- (swap X)

0 1 2 3 4   0 1 2 3 4
great   great
↓ smaller a   ↓ smaller b

```
if( solve( smaller a (gr), smaller b (gr)) == True
    && solve( smaller a (eat), smaller b (eat)) == True)
```

then string a & string b is scrambled.

```
if( solve(a.substr(0,i), b.substr(0,i)) == True
    && solve(a.substr(i,n-i), b.substr(i,n-i)) == True)
```

→ if (case-I || case-II) → if any case will true
scrambled string

Base Condition -

```
if (a.length != b.length)
    return false;
if (both empty)
    return True;
if (a.compare(b) == 0)
    return True;
if (a.length() <= 1)
    return false;
```

Main Code-

```
int main()
{ string a
  string b
  cin>> c>> b
→ a.length != b.length => false
→ Both empty => true.
  Solve(a,b)
}

bool solve(string a, string b)
{ if (a.compare(b) ==0)
      return True
  if (a.length <=1)
      return false

  int n= a.length();
  bool flag = False;
for (int i=1; i<n; i++)
{
  if (core I || core II) {
      flag = true
      break
  }
}
return flag;
```

* Scrambled string memoization

unordered_map <string, ~~int~~ bool> mp;

Key	value
"a b"	true

```
bool solve( string a, string b)
{
  if (a.compare(b) == 0)
     return True;
  if (a.length() <= 1 )
     return False;
  string key = "a";
  key.push_back(' ');
  key.append(b);
  if(mp.find(key) != mp.end())
     return mp[key];
  int n = a.length();
  bool flag = false;
  for(int i=1; i<n; i++)
  { if (caseI || caseII){
      flag = true;
      break;
    }
  } mp[key] = flag;
  return flag;
}
```

* 7 Egg Dropping Problem - (In Worst Case)

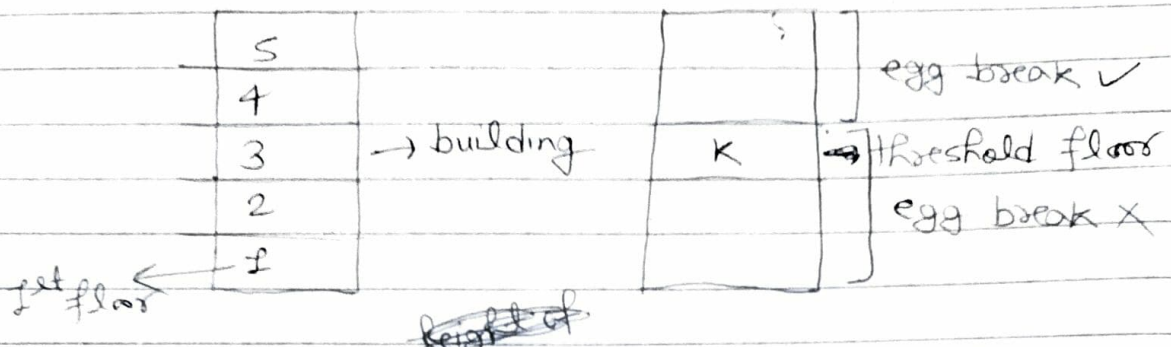
I/P

e = 3
f = 5

O/P

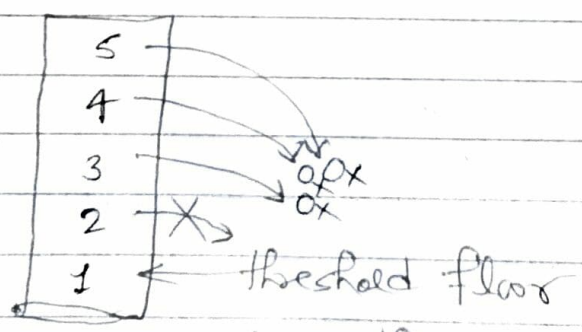
3

e : no of egg
f : no of floor



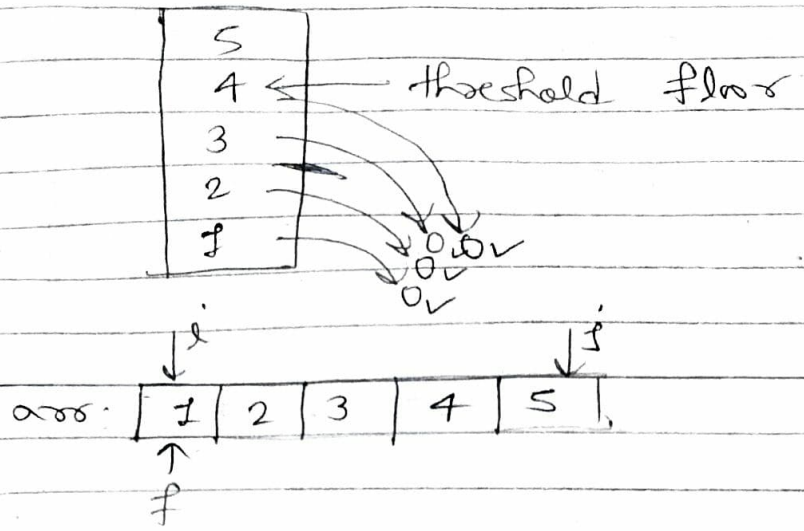
Find that ~~height of~~ threshold floor or critical floor with minimum no of attempts. ~~we have~~ in worst case.

Acc. to question we have only 3 eggs & we have to ~~write~~ wisely utilize of attempts so that we will find that threshold floor.



If we start throwing eggs from top then we can't find threshold floor.

So we have to start throwing from bottom.

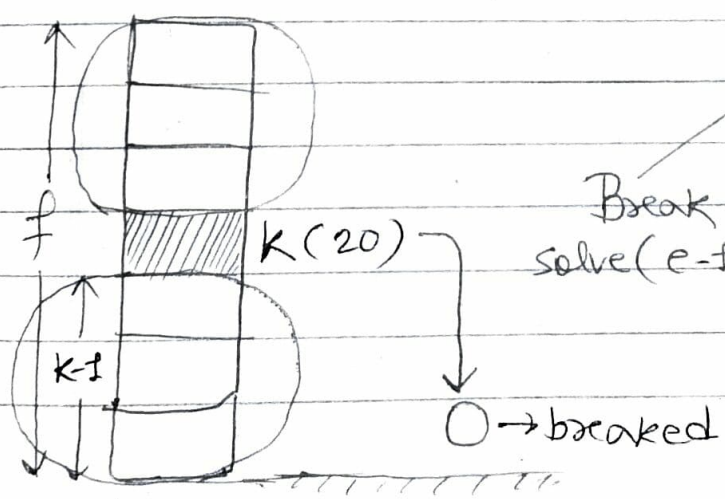


Base Condition-

```
if (e==1)
    return f;
if (f==0)
    return 0;
if (f==1)
    return 1;
```

```
if (f==0 || f==1)
    return f;
```

Approach-



Solve (e, f)

Break ✓
solve(e-1, k-1)

Break X
solve(e, f-k)

Code-

```
int solve(int e, int f)
{
    if (f==0 || f==1)
        return f;
    if (e==1)
        return f;
    int mn = INT_MAX;
    for (int k=1; k<=f; k++)
    {
        int temp = 1 + max( solve(e-1, k-1),
                            solve(e, f-k));
        mn = min(mn, temp);
    }
    return mn;
}
```

in worst case

Memoization-

```
int static t[11][51];   → Globally
memset(t, -1, sizeof(t));   → in main()

→ if (t[e][f] != -1)
      return t[e][f];

→ t[e][f] = mn;
```

Optimization

```
if (t[e-1][k-1] != -1)
    int low = t[e-1][k-1];
else {
    int low = solve(e-1, k-1);
        t[e-1][k-1] = low; }

if (t[e][f-k] != -1)
    int high = t[e][f-k];
else {
    int high = solve(e, f-k)
        t[e][f-k] = high; }
```