Point To Remember

** Set method can be used in Subset Sum-II problem also, it can be used when we have duplicacy in array.

** Problem with set is that time complexity increases.

(XII) Rope Cutting Problem: $O(3^n)$

Given rope of length n, cut the rope in maximum no. of pieces, piece length can be a, b or c.

eg:

I/P → n=5, a=2, b=5, c=1

o/p ⇒ 5 (1,1,1,1,1) → 5 ropes each of length 1(c)

$0 < a,b,c \leq n$

```
int MaxPieces(int n, int a, int b, int c)
{
    if(n==0) return 0;
    if(n<0) return -1;

    int ans = max(max(MaxPieces(n-a, a, b, c),
                      MaxPieces(n-b, a, b, c)),
                      MaxPieces(n-c, a, b, c));

    if ans == -1
        return -1;
    return ans+1;
}
```

max(max(a,b),c); ← finding max of 3 no.

→ Observe Carefully → we are writing this bz when all the three function call return -1, (ans+1) will convert it into 0 which may be converted to 1 giving us an wrong answer.

-1 -1 -1

-ve -ve -ve

max(-1,-1,-1) = -1

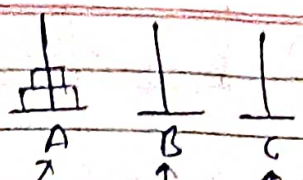
ans+1 = -1+1 = 0

Wrong

$T(n) = 1$ for $n=1$

$T(n) = 2T(n-1)+1$

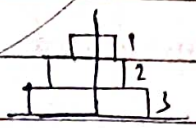
(XIII) **Tower of Hanoi** $O(2^n-1) == O(2^n)$



return the count of movement required.

Source Auxillary target

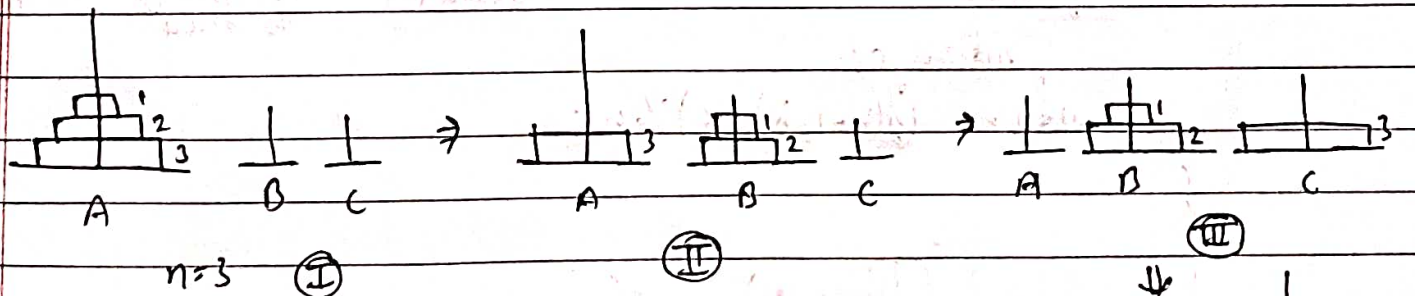
```
int toh(int n, char A, char B, char C)
{
    if(n==1)
    {
        cout<< "Move 1 from" <<A << "to" << C <<endl;
        return 1;
    }
    int l = toh(n-1, A, C, B);
    cout<<"Move" <<n <<" from" <<A<<" to" <<C<<endl;
    int r = toh(n-1, B, A, C);
    return l+r+1;
}
```



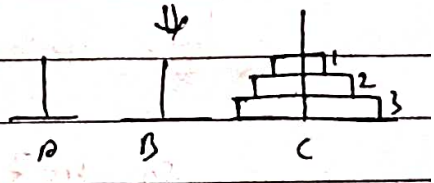
variable stand 'B' character

Logic *

eg : n=3



→ in 2nd step we are moving (n-1) rod from A to B using C as an auxillary. ~~to~~



→ in 3rd step we simply move the nth rod from A to C always

→ in 4th step we again move (n-1) rods from B to C using A as auxillary.

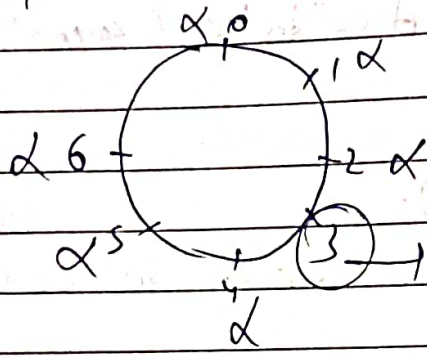
Po

(XIV) **Josephus Problem** (TC → O(n))

→ There are n people standing in a circle we have to kill k^{th} person in every iteration

→ zero based indexing if n=7 people 0→6

eg I/P → n=7, k=3
O/P → 3 → index

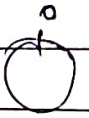


in main
cout << f(n,k)+1;

→ for getting position simply write ~~return~~ +1 in main function call of this function

Code returns index

```
int f(int n, int k)
{
    if(n==1)
        return 0;
    return (f(n-1,k)+k)%n;
}
```



→ return 0 → when only one person is there

logic behind $(f(n-1,k)+k)\%n$

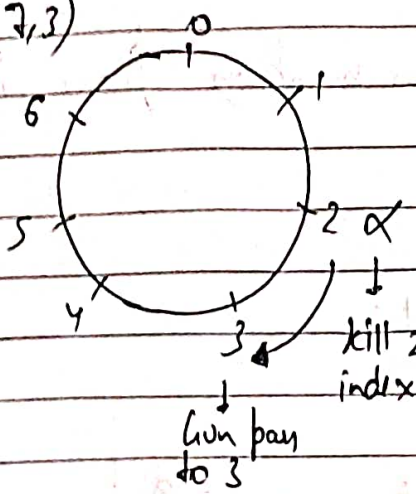
say n=7, k=3
i.e. f(7,3) call f(6,3) → f(n-1, k)
0-6 0-5

→ जब next function call होगा, तब f(n-1,k) एक नया function call होगा उसका पुराने से लेना देना नहीं।

→ नए function का numbering 0 → 5 होगा

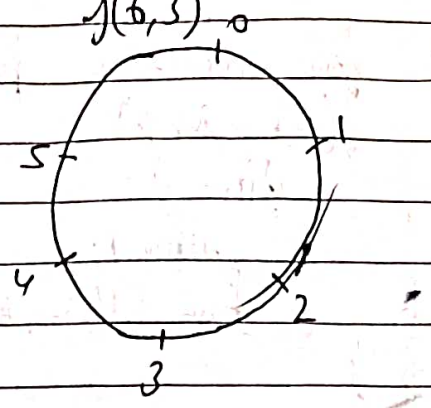
दो नया रहा है (size)

$J(7,3)$



$\Rightarrow$

$J(6,3)$



अब इस function call में फिर से 0 से start होगा, लेकिन हम चाहते 3 से start हो

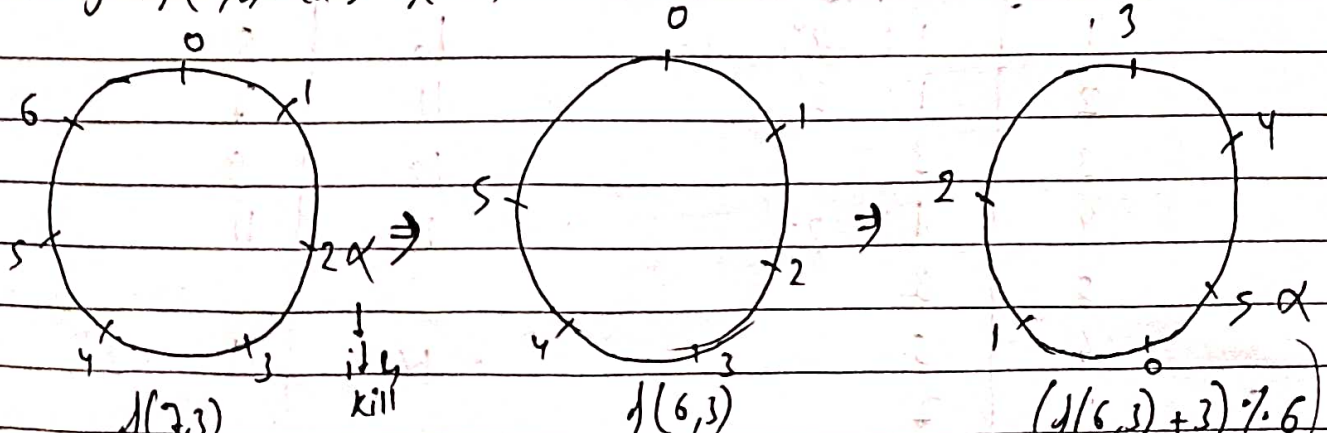
See carefully $J(6,3)$ में हम चाहते हैं ^ 0 → 3 बन जाए

0	3
1	4
2	5
3	0
4	1 → id 4 बनाना है
5	2

id 4 है

$\therefore$ ~~$J(n-1,k)+k)\%n$~~ $(J(n-1,k)+k)\%n$

say $J(7,3)$ calls $J(6,3)$



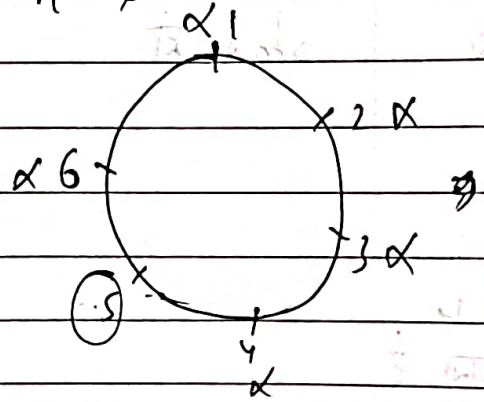
More in Josephus Problem:

(1) for returning position when numbering from 1→n

```
J(int n, int k)
{  if(n==1)
      return 1;
   return (J(n-1,K)+K-1)%n+1;
}
```

(2) N people in circle, we kill every 2nd person till only one person is left

eg: n = 6



1 kills 2
3 " 4
5 " 6
1 " 3
5 " 1

5 remain

Ans → 5

I/P → n=6 O/P → 5

n	W(n) → winning position	
1	1	2^0
2	1	2^1
3	3	
4	1	2^2
5	3	
6	5	
7	7	
8	1	2^3
9	3	
10	5	
11	7	
12	9	
13	11	
⋮	1	
16	1	2^4

(i) In Josephus Equation every even position dead
(ii) if $n = 2^a$ a → 0 to any no. then $W(n) = 1$

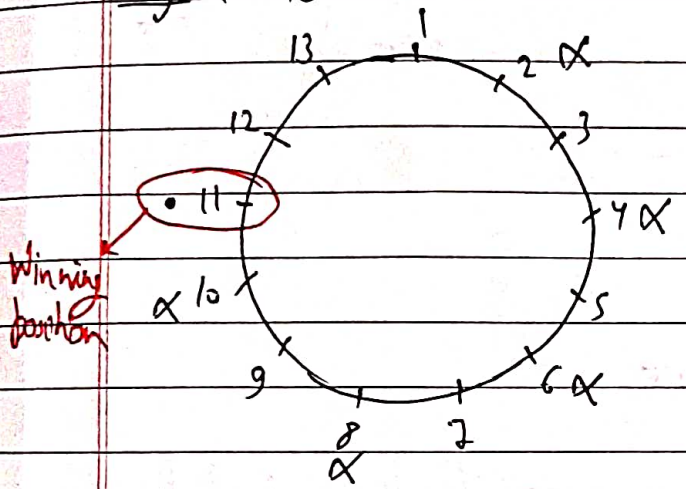
(a) If we write
$n = 2^a + l$
then $W(n) = 2l + 1$

eg: $n = 19$
$n = 2^4 + 3$
$W(n) = 2 \times 3 + 1 = 7$

$n = 13$
$n = 2^3 + 5$
$W(n) = 11$

↓
after l kills whoever turns is the winning position

eg: $n = 13$



$13 = 2^3 + (5)$
↓
after 5 kills whoever turns is a winning position

(b) $n = 41 = 101001$
$W(n) = 000011$
$= 16 + 2 + 1$
$= 19$ Ans

→ Giving n find W(n)

(3) To find leftmost bit set position.
I/P ⇒ 13 → 1101
3 2 1 0
O/P ⇒ 3

floor(log2(n));

standing on

~~(3) Given 1 to n numbers, people die in multiple of 2,4~~

(4) Lucky Numbers

Given n tell will it b survive.

I/P → n=13

n=13, i=2 | 1 2 3 4 5 6 7 8 9 10 11 12 13 | $\frac{13}{2} = 6$ person died

every 2nd no. die

n=7, i=3 | 1 3 5 7 9 11 13 | $\frac{7}{3} = 2 \Rightarrow 7-2 = 5$ person alive

every 3rd no. die

∴ alive = 13 − 6 = 7

n=5, i=4 | 1 3 7 9 13

every 4th no. die

n=4, i=5 | 1 3 7 13

when n<i stop & return true

if (n%i) == 0 return false means it died in between

Code

```
bool f(int n, int i)        // 2 → initial value of i
{
    if(n%i==0)
        return 0;
    if(n<i)
        return 1;
    return f(n-(n/i), i+1);
}
```

n-(n/i) ↑ no. of alive person after 1st elimination & so on.

Pow(x,n)

(XV) Binary Exponentiation — O(log n)

implement pow(x,n) which calculates x raised to the power n ie (x^n)

n is int $-2^{31} <= n <= 2^{31}-1$ (n can be negative)

x can be decimal ie x is double

$10^{-4} <= x^n <= 10^4$

last log value x^n mod have to use

ans can be decimal ∴ double

$x^{-n} = \frac{1}{x^n}$

calculate x^n at last divide by 1

```
double Pows(double x, int N)
{
    long long n = N;
    double ans = 1;   or ans = 1.0;  (same वात)
    if(n < 0)
        n = -1 * n;
    while(n != 0)
    {
        if(n % 2 == 0)
        {
            x = x * x;
            n = n/2;
        }
        else
        {
            ans = ans * x;
            n = n-1;
        }
    }
    if(N < 0)
        ans = 1/ans;
    return ans;
}
```

I/P → x = 2.00000, n = 10

O/P → 1024.00000

logic

~~$(1^2)^{21} \rightarrow x^n$~~

~~= n odd ∴~~

~~if x is e~~

① if n is even n = 10

$(2)^{10} = (4)^5$

x x*x

$x = x * x$ & $n = n/2$

② if n is odd n = 7

$(2)^7 = 2.(2)^6$

n = n-1

③ power odd आया तो even होते ही step it at even हो जाएगा।

④ at very last n will be 1 at that step

ans = ans * x & n becomes 0

eg: 3^7 $x=3, n=7$

→ $3 \cdot 3^6$

ans = 3, do calculation for 3^6

→ $(3.3)^3 = (9)^3$, $x=9$

→ $9.(9)^2$

↳ ans = ans * x

= 3 * 9 = 27

→ do for $(9)^2$

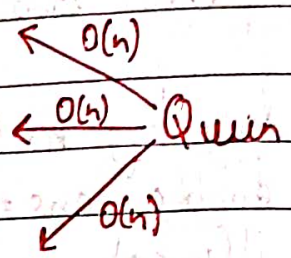
$(9.9)^1 = (81)^1$, x

→ $81.(81)^0$

n = 0

↳ ans = 27 * 81

= 2187



XVI N-Queen Problem

Given (n×n) chess board

place n queens on n×n chessboard such that no two queen attack each other.

logic say 4×4 n=4

col →

	0	1	2	3
0	Q			
1			Q	
2		Q		
3				

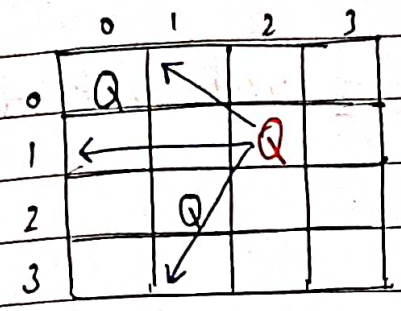
vector string → v

→ We move col+1 →

→ We have to check that it is safe to put Queen at v[0][0], v[2][3] etc.

→ To check if position v[i][j] safe or not we have to check in only three direction

already present



→ if any Queen comes already in the way then we can't put Queen at that position.
→ here v[1][2] is unsafe.

→ when col=2 we check for row → 0 to 3 in which it is safe.

Approach - I :

```
class Solution{
public:
    bool issafe(int row, int col, vector<string> board, int n)
    {
        int nrow=row, ncol=col;
        while(row>=0 && col>=0)  ⟶ for upper diagonal check (↖)
        {
            if(board[row][col]=='Q') return false;
        } row--; col--; }
```

* not do board[row-1][col-1] bz when row=0 (row-1) can't exist.

```
        while( row=nrow; col=ncol;
        while(col>=0)  → for column (←)
        {
            if(board[row][col]=='Q') return false;
            col--;
        }
        row=nrow; col=ncol;
        while(row<n && col>=0)
        {
            if(board[row][col]=='Q') return false;
            row++;
            col--;
        }
    return true;
```

```
public:
void f(int col, vector<string> &board, vector<vector<string>>
    &ans, int n)
{
    if(col==n)
    {
        ans.push_back(board);  → all possible solutions for particular n is pushed.
        return;
    }
    for(int row=0; row<n; row++)
    {
        if(issafe(row, col, board, n))
        {
            board[row][col] = 'Q';
            f(col+1, board, ans, n);
            board[row][col] = '.';
        }
    }
}

public:
    vector<vector<string>> Solve(int n){

        vector<vector<string>> ans;
        vector<string> board(n);
        string s(n,'.');
        for(int i=0;i<n;i++)
            board[i] = s;
        f(0, board, ans, n);
        return ans;
    }
};
```

→ Say n=4

s = ...
    0123

here board is

....
....
....
....

Approach (II) Hashing

```
public:
void f(int col, vector<string> &board, vector<int> &lrow, vector<int>
    &ldiag, vector<int> &udiag, vector<vector<string>> &ans, int n)
{
    if(col==n)
    {
        ans.push_back(board);
        return;
    }
    for(int row=0; row<n; row++)
    {
        if(lrow[row]==0 && ldiag[row+col]==0 &&
           udiag[n-1+col-row]==0)
        {
            board[row][col]='Q';
            lrow[row]=1;
            ldiag[col+row]=1;
            udiag[n-1+col-row]=1;
            f(col+1, board, lrow, ldiag, udiag, ans, n);
            board[row][col]='.';
            lrow[row]=0;
            ldiag[row+col]=0;
            udiag[n-1+col-row]=0;
        }
    }
}
public: vector<vector<string>> Solve(int n){
    vector<vector<string>> ans;
    vector<string> board(n);
    vector<int> lrow(n,0), ldiag(2*n-1,0), udiag(2n-1,0)
    string s(n,'.');
    for(int i=0; i<n; i++) board[i]=s;
    f(0, board, lrow, ldiag, udiag, ans, n);
    return ans;
}
```

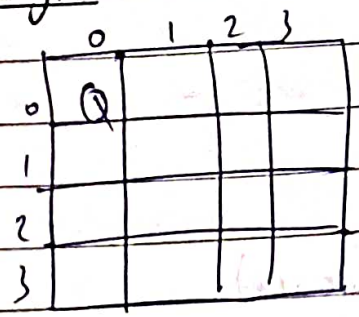
→ marking 1 means Queen at present

→ unmark b/z backtracking

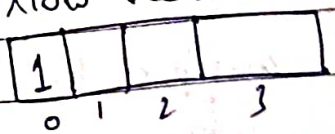
lleft = left row (←)
ldiag = lower diagonal (↙)
udiag = upper diagonal (↖)

Hashing logic

lrow

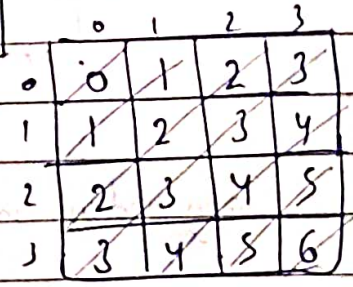


lleft vector (n=4)

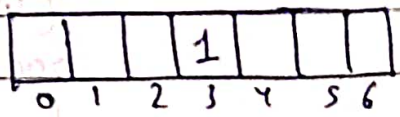


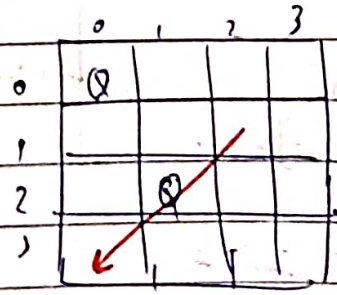
when col+1 = 1
since lrow[row] = 1 we can't put at [0][1] location
now++ row=1
lrow[1] == 0
∴ we can put at [1][1]

ldiag



→ Size of ldiag vector = 2n-1
for n=4 = 7





→ We have to check can we put Queen at [1][2]

→ here we can see [2][1] is taken means in hash function it is marked 1
ldiag[2+1] = ldiag[3]

→ When we go to check for ldiag[1+2] = ldiag[3] ≠ 0
— Queen can't come at [1][2]

udiag

n=4 [n-1 + col - row]

	0	1	2	3
0	3	4	5	6
1	2	3	4	5
2	1	2	3	4
3	0	1	2	3

3

Similarly this also.

* When all three hash function are true then only we can put Queen at that position.

String

① s.insert(index, str) ← string

eg: s = "Hi"
str = "RAJ"
s.insert(2, str)
o/p → HiRAJ

② to_string(n) ← int

→ int to string

③ stoi(str)
int n = stoi(str);
↓ ↑ string
string to int

(XVII) Catalan Sequence : → used in BST

→ Catalan numbers are a sequence of natural numbers

$$C_n = \frac{1}{n+1} {}^{2n}C_n = \sum C_i C_{n-i}, \quad i = 0 \text{ to } n-1$$

already defined →

$C_0 \rightarrow 1$
$C_1 \rightarrow 1$
$C_2 \rightarrow C_0C_1 + C_1C_0 = 2$
$C_3 \rightarrow C_0C_2 + C_1C_1 + C_2C_0 = 5$
$C_4 \rightarrow C_0C_3 + C_1C_2 + C_2C_1 + C_3C_0 = 14$

$$C_n = \frac{2n!}{(n+1)!\,n!}$$

→ Mathmatical formula

→ This help in writing Recursive code

Jibonacci ~~fibnocci~~ Sequence की तरह concept है

Code

```
int f(int n)
{
    if(n<=1)
        return 1;
    int ans=0;
    for(int i=0;i<n;i++)
    {
        ans += f(i) * f(n-i-1);
    }
    return ans;
}
```