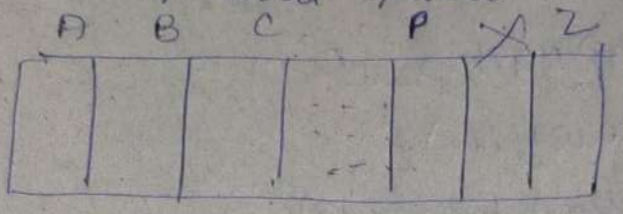
Binary Search:

-> Consider a dictionary and you want to search word "mediate"



-> You have selected the page by random and it is P.

-> You know mediate will be ~~there~~ before P.

-> So we will concencentrate on the left half and discard the right half.

-> Similarly the process will be continued till the word mediate is found, If no mediate" then ,it should return -1" indicating no such word.

Program:

```
int left=0;
int right=arr.length-1;
while(left<=right){
        int mid=(left+right)/2;
    if(arr[mid]==data)
            return mid;

    else if(arr[mid]<data)
            left=mid+1;

    else
            right=mid+1;
}
return-1;
```

Note; If the Array is big. , so in order to maintain the mid value in Integer ,

$$mid = left + \frac{(right-left)}{2}$$

16 Broken Economy:

Input: 10-n

1
5
10
15
22
33
40
42
55
66
34 → d

(5, 10, 15, 22, 33, 40, 42, 55, 66 — n elements)

Output:
40
33

Constraints:
$1 <= n <= 1000$
$-10^9 <= n_1, n_2 \ldots n$ elements $<= 10^9$
$-10^9 <= d <= 10^9$

Steps:

1. Given the inputs n, array elements and d
2. As per the problem statement, we need to display the ceil and floor of demonination if d is not present else if d is present; print d.
3. So, initially we will consider ceil = -1, floor = -1
4. Now, we will perform the binary search.
5. If $d < arr[mid]$, so we can think at this point ceil = arr[mid] and we will search in the first half.
6. If $d > arr[mid]$, so we can think at this point floor = arr[mid], and we will search in the second half.
7. If $d == arr[mid]$, we can return the arr[mid].
8. We will continue the process till left <= right

Dry Run:

n=10, ceil=-1, Floor=-1, data=34

0	1	2	3	4	5	6	7	8	9
1	5	10	15	22	33	40	42	55	66

left=0,
right=10-1=9, 0<=9 ✓

mid = (0+9)/2 = 4

a[4]=22 <= 34,

floor = a[4] = 22

left = 4+1 = 5

5<=9 ✓

mid = (5+9)/2 = 7,

a[7] = 42 > 34

ceil = a[7] = 42

right = 7-1 = 6

5<=6 ✓

mid = (5+6)/2 = 5

a[5] = 33 < 34

floor = a[5] = 33

left = 5+1 = 6

6<=6 ✓

mid = (6+6)/2 = 6

a[6] = 40 > 34

ceil = a[6] = 40

left = 4+1 = 3

3<=6, mid = 4

a[4] = 22 < 34

floor = a[4] = 22

left = 3-1 = 2

Rough work

6 <= 6

mid = 6+6/2 = 6

a[6] = 40 > 34

ceil = a[6] = 40

right = 6-1 = 5

6 <= 5 X ⟶

Program;

```
int n = sc.nextInt();
int a[] = new int[n];
for(int i=0; i<n; i++)
        a[i] = sc.nextInt();
int d = sc.nextInt();
int left = 0;
int right = a.length-1;
int ceil = -1;
int floor = -1;
while(left <= right){
        int mid = (left+right)/2;
    if(d < a[mid]){
            ceil = a[mid];
            right = mid-1;
    }
    else if(d > a[mid]){
            floor = a[mid];
            left = mid+1;
    }
    else{
        System.out.println(a[mid]);
        return;
    }
}
System.out.println(ceil);
System.out.println(floor);
```

19. First Index and Last Index:

Input:

15 → n

1
5
10
15
22
33
33
33
33
33
40
42
55
66
77

elements

33 — d

Output:

5
9

Constraints:

1 <= n <= 1000

1 <= n1, n2 .. n elements <= 100

1 <= d <= 100

Steps:

1. Takes the inputs n, elements and d.
2. As per the problem statement, we need to print the first occurence and last occurence of d.
3. So, we will use binary search to get the mid element, if ar(mid) == d, we will go towards the left side in order to get the first occurence by discarding the right side.

4° Similarly, we will run another binary search to get the mid element, if arr[mid] == d, we will go towards the right side in order to get the last occurence by discarding the left side.

Dry run:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	5	10	15	22	33	33	33	33	33	40	42	55	66	77

First Binary Search

fi = -1, d = 33
left = 0, right = 15-1 = 14

0 < = 14 ✓
mid = 7, a[7] = 33 == 33 ✓
fi = 7, right = 7 - 1 = 6

0 < = 6 ✓
mid = 3, a[3] = 15 < 33
left = 3+1 = 4

4 < = 6 ✓
mid = 5, a[5] = 33 == 33
(fi = 5) right = 5-1 = 4

4 < = 4
mid = 4, a[4] = 22
22 < 34
left = 4+1
5 < = 4 ✗
(fi = 5)

Second Binary Search

li = -1, d = 33
left = 0, right = 15-1 = 14

0 < = 14 ✓
mid = 7, a[7] = 33 = 33 ✓
ri = 7, left = 7+1 = 8

8 < = 14 ✓
mid = 11, a[11] = 42 > 34
right = 11-1 = 10,

8 < = 10 ✓
mid = 9, a[9] = 33 = 33
(ri = 9), left = 9+1 = 10

10 < = 10 ✓
mid = 10, a[10] = 40 > 33
right = 10 - 1 = 9

10 < = 9 ✗

(ri = 9)

Program:

```
int n = sc.nextInt();
int a[] = new int[n];
for (int i = 0; i < n; i++)
        a[i] = sc.nextInt();
int d = sc.nextInt();
binarySearch(a, d);
```

19 Public static void binarySearch (int arr[], int d) {

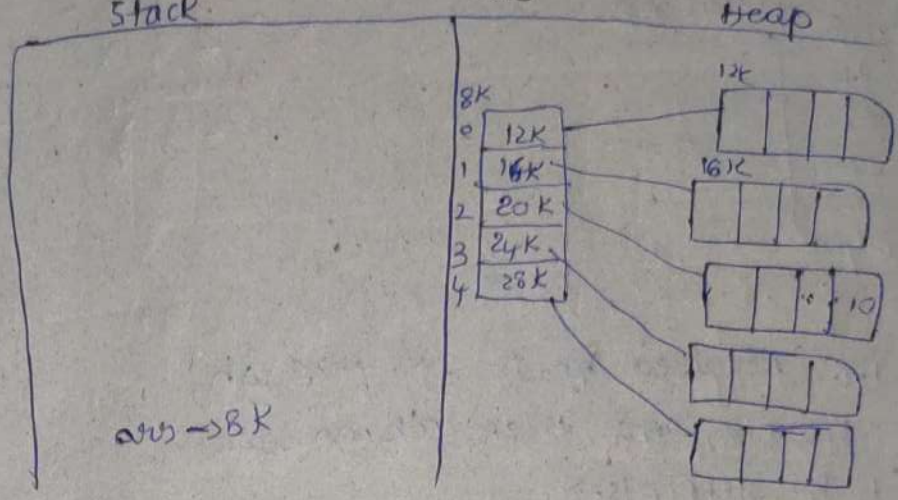
```
int fi = -1;
int left=0;
int right=arr.length-1;                    // First occurr
while (left<=right) {                      (searching
    int mid=(left+right)/2;                left side)
    if (arr[mid]==data){
        fi=mid;
        right=mid-1;
    }
    else if (arr[mid]>data)
        right=mid-1;
    else
        left=mid+1;
}

left=0;
right=arr.length-1;
int ri=-1;
while (left<=right) {                      // Last occurren
    int mid=(left+right)/2;                (searching in
    if (arr[mid]==data){                   right side)
        fi=mid;
        left=mid+1;
    }
    else if (arr[mid]>data)
        right=mid-1;
    else
        left=mid+1;
}
```

S
1.
2.
3

2D Arrays:

```
int[][] arr; //Declaring
arr = new int[5][4];
```

rows ↑ columns ↑



arr[2][3] = 10

arr[2][3] } Access time is
arr[0][0] } equal.

No of rows → arr.length
No of columns → arr[i].length

```
int[][] arr = {{5,1,2}, {3,4,5}, {6,7,8}};
```

→ Giving column declaration is not mandatory.

```
int[][] arr = new int[5][];
arr[0] = new int[4];
arr[1] = new int[3];
arr[2] = new int[5];
arr[3] = new int[1];
arr[4] = new int[2];
```

1. 2d Arrays Demo

Input:

2 — rows (n)
2 — columns (m)
4
11
12
13
14
21
22
23
24

(11 12 13 14 / 21 22 23 24 — elements)

Output:

11 12 13 14
21 22 23 24

Constraints;

$1 <= n <= 10^2$
$1 <= m <= 10^2$
$-10^9 <= c_1, c_2 .. <= 10^9$
n x m elements

Steps;

1. Take the Required inputs, go through each rowwise and then column wise print the elements.

Program;

```
int rows = sc.nextInt();
int cols = sc.nextInt();
int[][] a = new int[rows][cols];
for(int i=0; i<rows; i++) {
    for(int j=0; j<cols; j++) {
        a[i][j] = sc.nextInt();
        print[a[i][j] + " ");
    }
println();
}
```

2. Matrix Multiplication;

Input;

```
2   → n1 → (rows)
3   → m1 → (columns)
10
0
0
0
20
0
3   → n2 → (rows)
4   → m2 → (columns)
1
0
10
0
0
1
1
2
1
1
0
0
```

first Matrix elements

second matrix elements

Output;

```
10  0  10  0
0   20 20  40
```

Constraints

$1 <= n1 <= 10^2$

$1 <= m1 <= 10^2$

$-10^9 <= e11, e12 \ldots$ $n1 * m1$ elements $<= 10^9$

$1 <= n2 <= 10^2$

$1 <= m2 <= 10^2$

$-10^9 <= e11, e12, \ldots n2 * m2$ elements $<= 10^9$

Steps;

1. To perform the matrix multiplication, first we need to check $m1 == n2$ [columns of first matrix and rows of second matrix should be equal], if not equal the matrix multiplication cannot be performed.
2. The resultant size of matrix is $(n1 \times m2)$
3. Consider Example;

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 20 & 20 & 40 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & \\ 1 & 2 & 1 & \\ 1 & 0 & 0 & \end{bmatrix}$$

2×3 $\quad$ 3×4

[Wrong Example]

3. Consider Example

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 20 & 20 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

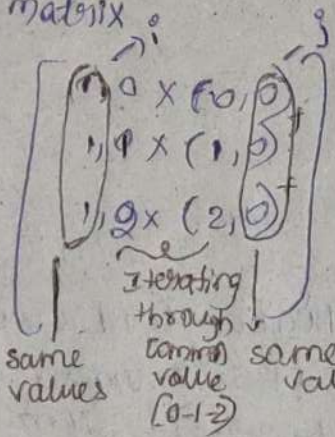
$2 \times 3 \quad 3 \times 4$

Now consider to get the first element of the resultant matrix

$10 \times 1 + 0 \times 0 + 0 \times 1 = 10$

$(0,0) \times (0,0) + (0,1) \times (1,0) + (0,2) \times (2,0)$ (0 0)

→ we observe that dashed lines are having same values, we check for element example: (1,0) of resultant matrix.



→ So we will fix the outer loop for iterating through rows of resultant matrix. (i)

→ So, we will take another loop j covering the columns. (j)

→ In each iteration of column, we will take Ans = 0

→ Then we will take another loop which the covers the values of common value (k)

→ We will get the sum as,

```
ans += a1[i][k] * a2[k][j];
```

-> We will store the resultant value in the respective position of Resultant Matrix.

Dry Run:

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 20 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

2x3 3x4

int c[][]=new int[m1][m2] , m1=2
m2=4

common value = m2/m1

i=0, i<2 ✓

j=0, j<4

ans=0

k=0, k<m2 , 0<3 ✓

ans = 0 + a1[0][0] * a2[0][0]
= 0 + 10 * 1 = 10

k=1, k<3 ✓

ans = 10 + a1[0][1] * a2[1][0]
= 10 + 0 * 0 = 10

k=2, 2<3 ✓

ans = 10 + a1[0][2] * a2[2][0]
= 10 + 0 * 1 = 0

ans = 10 ✓ k=3<3 ✗

c[0][0] = 10

j=1, 1<4

ans = 0.

k=0, k<3

ans = 0 + a1[0][0] * a2[0][1]
= 0 + 10 * 0 = 0

k=1, 1<3 ✓

ans = 0 + a1[0][1] * a2[1][1]
= 0 + 0 * 1 = 0

k=2, 2<3 ✓

ans = 0 + a1[0][2] * a2[2][1]
ans = 0 + 0 * 1 = 0

k=3<3 ✗

c[0][1] = 0

j=2<4

ans=0

k=0<3 ✓

ans=0+a1[0][0]xa2[0][2]

=0+10x1=10

k=1<3 ✓

ans=10+a1[0][1]xa2[1][2]

=10+0x1=10

k=2<3 ✓

ans=10+a1[0][2]xa2[2][2]

=10+0x0=10 , k=3<3 X

c[0][2]=10

j=3<4

ans=0

k=0<3 ✓

ans=0+a1[0][0]xa2[0][3]

=0+10x0=0

k=1<3 ✓

ans=0+a1[0][1]xa2[1][3]

=0+0x2=0

k=2<3

ans=0+a1[0][2]xa2[2][3]

=0+0x0=0

k=3<3X

c[0][3]=0

i=1<2 ✓

j=0<4 ✓

ans=0

k=0<3 ✓

ans=0+a1[1][0]xa2[0][0]

=0+0x1=0

k=1<3 ✓

ans=0+a1[1][1]xa2[1][0]

=0+20x0=0

k=2<3 ✓

=0+a1[1][2]xa2[2][0]

ans = 0+0x1 = 0 k=3<3 ✗

c[1][0] = 0

j=1<4 ✓

ans = 0

k=0<3 ✓

ans = 0 + a1[1][0] x a2[0][1]

= 0 + 0x0

= 0

k=1<3 ✓

ans = 0 + a1[1][1] x a2[1][1]

= 0 + 20x1 ~~+0x1~~

ans = 20

k=2<3 ✓

ans = 20 + a1[1][2] x a2[2][1]

= 20 + 0x1 = 20

k=3<3 ✗

c[1][1] = 20

j=2<4 ✓

ans = 0

k=0<3 ✓

ans = 0 + a1[1][0] x a2[0][2]

= 0 + 0x1 = 0

k=1<3 ✓

ans = 0 + a1[1][1] x a2[1][2]

= 0 + 20x1 = 20, k=2<3 ✓

ans = 20 + a1[1][2] x a2[2][2]

= 20 + 0x0 = 20

k=3<3 ✗

c[1][2] = 20

j=3<4 ✓

ans = 0

k=0<3 ✓

ans = 0 + a1[1][0] x a2[0][3]

= 0 + 0x0 = 0

k=1<3 ✓

ans = 0 + a1[1][1] x a2[1][3]

= 0 + 20x2 = 40

k=2<3 ✓

ans = 40 + a1[1][2] x a2[2][3]

= 40 + 0x0 = 40

K=3L3X

c[i][3]=40

Program:

```
int n1= sc.nextInt();
int m1= sc.nextInt();
int a1[][]= new int[n1][m1];
for(int i=0; i<n1; i++){
        for(int j=0; j<m1; j++){
            a1[i][j]=sc.nextInt();
        }
}
int n2= sc.nextInt();
int m2= sc.nextInt();
int[][] a2 = new int[n2][m2];
for(int i=0; i<n2; i++){
        for(int j=0; j<m2; j++)
            a2[i][j]=sc.nextInt();
}
if(m1!=n2){
        println("Invalid input");
        return;
}
else{
    int c[][]=new int[n1][m2];
        for(int i=0; i<n1; i++){
            for(int j=0; j<m2; j++){
                int ans=0;
                for(int k=0; k<n2; k++){
                    ans+= a1[i][k]* a2[k][j];
                }
                c[i][j]=ans;
            }
}
for(int i=0; i<n1; i++){
        for(int j=0; j<m2; j++)
            print(c[i][j]+" ");
    println();
}
```

3. The State of Wakanda-1

Input: 3 - n rows
4 - m columns
11
12
13
14
21
22
23
24
31
32
33
34
41
42
43
44
elements

Output: 11
21
31 41
32 42
22
12
13
23
33 43
34 44
24
14

Constraints:

$1 <= n <= 10^2$

$1 <= m <= 10^2$

$10^9 <= e1, e2 ... n$ elements $<= 10^9$

Steps:

1. Take the inputs as per requirements.
2. 11 → 12 → 13 → 14
   ↓ ↑ ↓ ↑
   21 → 22 → 23 → 24
   ↓ ↑ ↓ ↑
   31 → 32 → 33 → 34
   ↓ ↑ ↓ ↑
   41 → 42 → 43 → 44

   As per the transition, we observe that in even column numbers, elements are traversing up-down and other column down-up.
3. Take outer loop that traverse through columns
4. If column number is even, traverse through up-down and print elements.
5. If column number is odd, traverse through down-up and print the elements.

Program;

```
for(int i=0; i<cols; i++){
    if (i%2==0){
        for(int k=0; k<rows; k++)
            Print(a[k][i]);
    }
    else{
        for(int k=rows-1; k>=0; k--)
            print(a[k][i]);
    }
}
```

4 Spiral Display

Input: 3 -n (rows)
5 -m (cols)

Output:

11
12
13
14
15
21
31
32
33
34
35
25
15
14
13
12
22
23
24

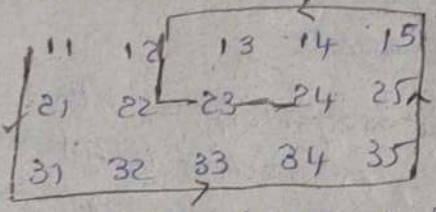
Constraints:

$1 <= n <= 10^2$

$1 <= m <= 10^2$

$-10^9 <= e1, e2 \ldots n*m\ elements <= 10^9$

Steps:

1. Take the inputs as per required.
2. 

through this

we can observe 4 stages: first col
last row
last col
first row

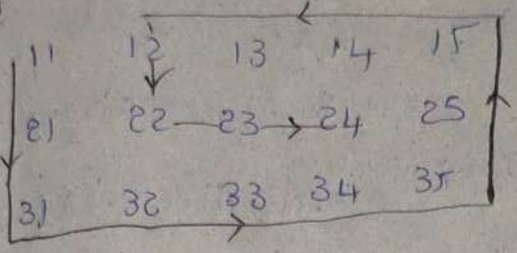
this pattern appears.

3. So, consider st row = 0,
st col = 0,
end row = rows - 1,
end col = cols - 1

4. After printing first col, do st col ++
5. After printing last row, do end row --;
6. After printing last col, do end col --.
7. After printing first row do st row ++
8. Continue the process till all elements are covered.

9. Check in the total numbers of elements are covered or not before performing the last col and first row operations.

Dry Run:



strow = 0
stcol = 0
endrow = 3-1 = 2
endcol = 5-1 = 4
count = 0,
total = 3*5 = 15

count < total.

0 < 15 ✓

i = strow, i <= endrow

i = 0, i <= 2 ✓

a(i)(stcol)) = [11].

c = 1

i = 1, i <= 2 ✓ a[i](stcol) = 21

c = 2

i = 2, i <= 2 ✓ a(i)(stcol) = 31

c = 3.

stcol = 0+1 = 1

i = stcol; i <= endcol

i = 1; i <= 4 ✓

a[endrow](i) = 32 ✓ → count = 4

i = 2, i <= 4 a[endrow](i) = 33 → count = 5

i = 3, i <= 4 a[endrow](i) = 34 → count = 6

i = 4, i <= 4 a[endrow](i) = 35 → count = 7

7 < 15 ✓ endrow = 2-1 = 1

i = endrow; i >= strow

i = 1, i >= 0

i = 1, i >= 0 ✓

a(i)(endcol) = 25 → count = 8

i = 0; i >= 0

a(i)(endcol) = 15 → count = 9

endcol = 4-1=3

9<15 ✓

i=endcol; i>=stcol

i=3; i>=1

print(a[strow][i]) = 14

count=10

i=2, i>=1

a[strow][i]=13 count=11

i=1, i>=1 a[strow][i]=12 count=12

strow=0+1=1

12<15 ✓

i=1, i<=2

print a[stcol][i]=22, cout 13

~~i=2, i<=2~~

~~a[stcol][i]=23; count 14~~

~~stcol~~ col=2

i=2, i<=3, i++

a[endrow][i]=23 count 4

3<=3, a[endrow][i]=24 count 15

15<15 ✗

Program:

```
int[][] a = new int[rows][cols];
int strow=0;
int stcol=0;
int endrow=rows-1;
int endcol=cols-1;
int total = rows*cols;
int count=0
while(count<total) {
        // left col
    for(int i=strow; i<=endrow; i++) {
            println(a[i][stcol]);
        count++;
    }
stcol++;
```

```
// bottom row
for(int i=stcol; i<=endcol; i++){
        System.out.println(a[endrow][i]);
    count++;
}
endrow--;

// last col
if(count<total){
    for(int i=endrow; i>=strow; i--){
        System.out.println(a[i][endcol]);
        count++;
    endcol--;
}
// first row
if(count<total){
    for(int i=endcol; i>=stcol; i--){
        System.out.println(a[strow][i]);
        count++;
    }
    strow++;

}
```

5. Ring Rotate

Input: 5 - rows
7 - columns

11	12	13	14	15	16	17	
21	22	23	24	25	26	27	
31	32	33	34	35	36	37	
41	42	43	44	45	46	47	
51	52	53	54	55	56	57	

Output: 2 - shells
3 - rotation

11	12	13	14	15	16	17
21	25	26	36	46	45	27
31	24	33	34	35	44	37
41	23	22	32	42	43	47
51	52	53	54	55	56	57

Constraints:

$1 <= n <= 10^2$

$1 <= m <= 10^2$

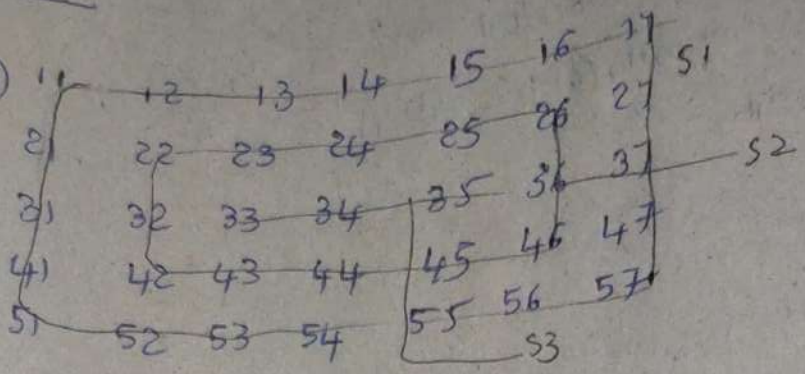
$-10^9 <= a11, a12 \ldots$

$n * m$ elements $<= 10^9$

$0 <= s <= \min(n, m)/2$

$-10^9 <= s <= 10^9$

Steps:

1) 

2) Given S:2

3) We will get S2 and we will store it in 1D Array

22	32	42	43	44	45	46	36	26	25	24	23

4) Given r=2 we will do 3 rotations

25	24	23	22	32	42	43	44	45	46	36	26

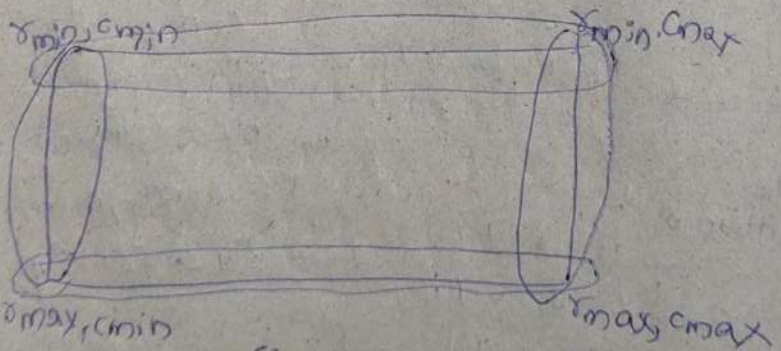
5) Then, we will back the 1D array into 2D Array

6) For getting the total size of a 1D Array,

For shell S:2 , $r_{min} = (1,1)$; c_{min}

So, $r_{min} = S-1$

$c_{min} = S-1$



Total size = $2((r_{max} - r_{min})+1) + 2((c_{max} - c_{min})+1) - 4$

Total size: $2(r_{max} - r_{min}) + 2(c_{max} - c_{min})$

Program:

```
int rows = sc.nextInt();
int cols = sc.nextInt();
int[][] a = new int[rows][cols];
for(int i=0; i<rows; i++) {
        for(int j=0; j<cols; j++)
                a[i][j] = sc.nextInt();
}
int s = sc.nextInt();
int r = sc.nextInt();
shellRotate(a, s, r);
display(a);
}
public static void shellRotate(int a[][], int s, int r)
{
int la = fill1Dfrom2D(a, s);
rotate(la, r);
fill2D(a, la, s);
}                                   // Getting the shell into 1D
public static int[] fill1Dfrom2D(int[][] a, int s) {
int rmin = s-1;
int cmin = s-1;
int rmax = a.length - s;
int cmax = a[0].length - s;
int sz = 2*(rmax - rmin) + 2*(cmax - cmin);
int[] la = new int[sz];
int index = 0;
//left
for(int i=rmin; i<=rmax; i++) {
    la[index] = a[i][cmin];
    index++;
}
```

```
        cmin++;
        // bottom
        for (int i = cmin; i <= cmax; i++) {
            la[index] = a[rmax][i];
            index++;
        }
        rmax--;
        // right
        for (int i = rmax; i >= rmin; i--) {
            la[index] = a[cmax][i];
            index++;
        }
        cmax--;
        // top
        for (int i = cmax; i >= cmin; i--) {
            la[index] = a[rmin][i];
            index++;
        }
        return la;
    }

    // Rotate the 1D
    public static void rotate(int[] la, int r) {
        r = r % la.length;
        if (r < 0) {
            r = r + a.length;
        }
        reverse(la, 0, la.length-1-r);
        reverse(la, la.length-r, la.length-1);
        reverse(la, 0, la.length-1);
    }
```

```
public static void reverse (int[] la, int left, int right) {
    while (left < right) {
        int temp = la[left];
        la[left] = la[right];
        la[right] = temp;
        left++;
        right--;
    }
}

// filling back 1D into 2D
public static void fill2D (int [][] a, int [] la, int s) {
    int rmin = s-1;
    int cmin = s-1;
    int rmax = a.length - s;
    int cmax = a[0].length - s;
    int index = 0;
    // left
    for (int i = rmin; i <= rmax; i++) {
        a[i][cmin] = la[index];
        index++;
    }
    cmin++;
    // bottom
    for (int i = cmin; i <= cmax; i++) {
        a[rmax][i] = la[index];
        index++;
    }
    rmax--;
    // right
    for (int i = rmax; i >= rmin; i--) {
        a[i][cmax] = la[index];
        index++;
    }
    cmax--;
```

```
//top
for(int i=cmax; i>=cmin; i--){
    a[rmin][i]=la[index];
    index++;
}
}
public static void display(int[][] arr){
    for(int i=0; i<arr.length; i++){
        for(int j=0; j<arr[0].length; j++){
            print(arr[i][j]+" ");
        }
        println();
    }
}
```

6. Exit Point of a Matrix:

Input:

4 → Rows
4 → Columns
0
0
0
1
0
0
0
0
0
0
0
0
0
1
0
1
0

Output:

1
3

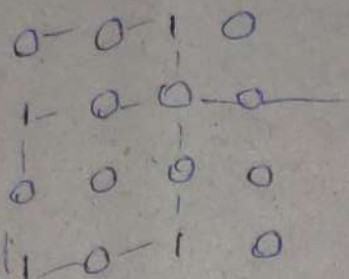
Constraints:

$1 <= n <= 10^2$

$1 <= m <= 10^2$

$e_1 e_2 \ldots n*m$ elements belong to the set $(0, 1)$

Steps:

1. 

2. Go through east, whenever we find 1 take 90° rotation.

3. Intially we take direction = 0
   we will consider d = 0 → For East
   = 1 → For South
   2 → For East
   3 → For North

4. We will change the value of dir. by
   dir = (dir + a[i][j]) % 4 ← represents 4 directions

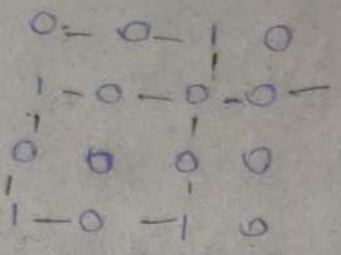
5. take i=0, j=0 based upon dir increment or decrement i, j values.

6. Continue the process, if i<0 do i++ → Break
   j<0 do j++ → Break
   i == a.length do i-- → Break
   j == a.length do j-- → Breaks

   because when the exit point occurs, indices should not be negative or greather than length.

7. At, the point Escape indexes respectively.

Dry Run:



i=0, dir=0

j=0

dir = (0 + a[i][j]) % 4

= (0+0) % 4

= 0

dir == 0,

j++ ; j = 1

i < 0 + j < 0 ✗ i == a.length j == a.length

dir = 0 + 0 % 4

= 0

j++ ; j = 2

i < 0 ✓ j < 0 i == a.length j == a.length

dir = (0 + 1) % 4

= 1

dir == 1

i++ ; i = 1

i < 0 ✗ j < 0 i == a.length j == a.length

dir = (1 + a[1][2]) = 1

dir == 1, i++ ; i = 2

dir = (1 + a[2][2]) = 1 + 0 = 1

dir == 1

i++ , i = 3

dir = (1 + a[3][2]) 1+1=2 2 % 4 = 2

dir == 2, j-- , j = 1

dir = (2 + 0) % 4 = 2

dir == 2 ✓ j-- , j = 0

dir = (2 + 1) % 4 = 3

dir = 3

i--, i = 2

dir = (3 + a[2][0]) % 4

= 3 % 4 = 3

dir = 3, i--, i = 1

dir = (3 + 1) % 4 = 0

dir = 0,

j++; j = 1

dir = (0 + 0) % 4 = 0

j++, j = 2

dir = (0 + 0) % 4 = 0

j++, j = 3

dir = (0 + 0) % 4 = 0

j++ ; j = 4

j == a.length ✓

so, j--;

j = 3

break

(i = 1, j = 3)

Program;

```
int dir = 0; //0 - e, 1 - s, 2 - w, 3 - n
int i = 0;
int j = 0;
while(true) {
    dir = (dir + a[i][j]) % 4;
    if (dir == 0)
        j++;
    else if (dir == 1)
        i++;
    else if (dir == 2)
        j--;
    else if (dir == 3)
        i--;
```

```
if(i<0){
    i++;
    break;
}
else if(j<0){
    j++;
    break;
}
else if(i==a.length){
    i--;
    break;
}
else if(j==a.length){
    j--;
    break;
}
}
println(i);
println(j);
```

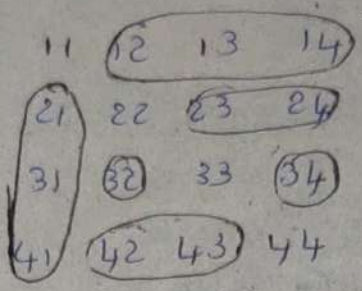
7. Rotate By 90 Degree;

Input;	output;	Constraints;
4 — n	41 31 21 11	1<=n<=10^2
11	42 32 22 12	-10^9<=e1,e2....2*n
12	43 33 23 13	elements<=10^9
13	44 34 24 14	
14		
21		
22		
23		
24		
31		
32		
33		
34		
41		
42		
43		
44		

Steps:

1. Take the inputs as per required

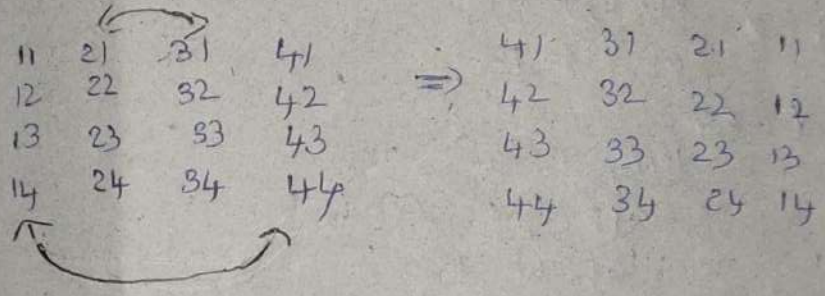
2. Eg: consider



3. First perform A^T,

11	21	31	41
12	22	32	42
13	23	33	43
14	24	34	43

4. Now, perform columns rearrangement



5. For perform Transpose, take outer loop i which traverse through array: length-1 and one more loop which is j=i+1 and traverse through array and swapping the elements

6. For column arranage, similar to reverse the array but we will be rearranging the columns.

Dry Run:

11 12 13 14

21 22 23 24

31 32 33 34

41 42 43 44

① A^T, $i = 0, i < 3$ [a.length-i]

$j = i+1 = 0+1 < 4$ [a.length]

temp = a[i][j] = 12

a[i][j] = a[j][i] = 21

a[j][i] = 12

$j = 1+1 = 2 < 4$

temp = a[i][j] = 13

a[i][j] = 31

a[j][i] = 13

$j = 3 < 4$

temp = a[i][j] = 14

a[i][j] = a[j][i] = 41

a[j][i] = 14

$j = 4 < 4$ ×

$i = 1, i < 3$

$j = i+1 = 1+1 = 2 < 4$

temp = a[i][j] = 23

a[i][j] = a[j][i] = 32

a[j][i] = 23

$j = 2+1 = 3 < 4$

temp = a[i][j] = 24

a[i][j] = 42

a[j][i] = 24

$j = 4 < 4$ ×

$i = 2, i < 3$ $j = 2+1 = 3 < 4$

temp = a[i][j] = 34

a[j][i] = 43

a[j][i] = 34

$j = 4 < 4$ ✗

$i = 3 < 3$ ✗

Now

11	21	31	41
12	22	32	42
13	23	33	43
14	24	34	43

② Rearranging columns

left = 0

right = a.length − 1 = 3

left < right = 0 < 3 ✓

row = 0; row < 4

temp = a[row][left] = 11

a[row][left] = a[row][right] = 41

a[row][right] = 11

row = 1 < 4

temp = a[row][left] = 12

a[row][left] = a[row][right] = 42

a[row][right] = 12

row = 2 < 4

temp = a[row][left] = 13

a[row][left] = a[row][right] = 43

a[row][right] = 43

row = 3 < 4

temp = a[row][left] = 14

a[row][left] = a[row][right] = 44

a[row][right] = 14

left = 0 + 1 = 1

right = 3 − 1 = 2

1 < 2 ✓

row = 0 < 4

temp = a[row][left] = 21

a[row][left] = a[row][right] = 31

a[row][right] = 21

row: 1 < 4
temp = a[row][left] = 22
a[row][left] = a[row][right] = 32
a[row][right] = 22

row: 2 < 4
temp = a[row][left] = 23
a[row][left] = a[row][right] = 33
a[row][right] = 23

row: 3 < 4
temp = a[row][left] = 33
a[row][left] = a[row][right] = 34
a[row][right] = 24

row: 4 < 4 ✗

left = 1 + 1 = 2
right = 2 - 1 = 1

2 < 1 ✗ ⇒

41	31	21	11
42	32	22	12
43	33	23	13
44	34	24	14

Program;

```
public static void rotateby90(int[][] a) {
    //transpose
for(int i=0; i<a.length-1; i++){
    for(int j=0; j<a.length; j++){
        int temp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = temp;
    }
}
```

```
// row column rearrangment

int left=0;
int right=a.length-1;
while(left<right){
    for(int row=0;row<a.length;row++){
        int temp=a[row][left];
        a[row][left]=a[row][right];
        a[row][right]=temp;
    }
    left++;
    right--;
}
}
```

8. The state of Wakanda-2

Input:

```
4
11
12
13
14
21
22
23
24
31
32
33
34
41
43
44
```

Output:

```
11
22
33
44
12
23
34
13
24
13.
```

Constraints:

$1 <= n <= 10^2$

$-10^9 <= e11, e12 \ldots n*m$ elements $<= 10^9$

Steps:

1.

```
11  12  13  14
21  22  23  24
31  32  33  34
41  42  43  44
```

diag0	diag1	dig2	dig3
(0,0)	(0,1)	(0,3)	(0,4
(1,1)	(1,2)	(1,3)	
(2,2)	(2,3)		
(3,3)			

2. We observe no of diagonals = n

3. We traverse through no of diagonal, we take i=0, j=diag and print respective value, increment the values till j<n.

Dry Run:

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

diag=0, 0<4 ✓

i=0, j=diag, 0<4

print(a[i][j])=11

i=1, j=1, 1<4

a[i][j]=22

i=2, j=2, 2<4

a[i][j]=33

i=3, j=3, 3<4

a[i][j]=44

i=4, j=4, 4<4 X

1<4 ✓

i=0, j=diag=1, 1<4

a[i][j]=12

i=1, j=2<4

a[i][j]=23

i=2, j=3<4

a[i][j]=24

i=3, j=3<4 X

2<4 ✓

i=0, j=2<4

a[i][j]=13

i=1, j=3<4 ✓

a[i][j]=24

i=2, j=4<4 X

3<4 ✓ i=0, j=3<4
a[i][j]=14
i=1, j=4<4 ✗

4<4 ✗

Program:

```
for(int diag=0; diag<n; diag++){
    for(int i=0, j=diag; j<n; j++)
        println(a[i][j]);
}
```

9. Saddle Price

Input: 4
11
12
13
14
21
22
23
24
31
32
33
34
41
42
43
43

Output: 41

Constraints:

$1 <= n <= 10^2$

$-10^9 <= e11, e12 \ldots n*m$ elements $<= 10^9$

Steps:

11	12	13	14
21	22	23	24
31	32	33	34
(41)	42	43	44

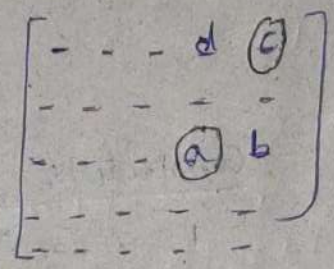
min in the row and max in the column

2. Traverse through the array, find minimum element of a row and its column number

3. Then check in that column if any number greater than the minimum number

If Yes → No saddle point
If No → print the Number

Why only one saddle point:



Assume there are two saddle points

If a is my ans,
$a < b$, $d < a$
$d < a < b$, $d < b$

If c is my ans,
$c < d$, $c > b$
$b < c < d$. $b < d$

Contradiction

So, Two Saddle Points are Not Possible.

Program:

```
for(int i=0; i<n; i++){
        //Traversing each row
    int min=0;
    for(int j=0; j<n; j++){
        if(a[i][j]<a[i][min]){
            min=j; //storing the minimum
                   value index
    }}

//checking that min value is
greater than any element in the matrix
```

```
boolean check=true;
for(int k=0;k<n;k++){
    if(a[k][min]>a[i][min]){
        check=false;
        break;
    }
// if no print the value, else go to next row
    if(check==true){
        println(a[i][min]);
        return;
    }
}
// No saddle price found
println("Invalid Input");
```

10. Search In A Sorted 2d Array.

Input:

4-n

11
12
13
14
21
22
23
24
31
32
33
34
41
42
43
44
43

elements

d

Output: 3
2

Constraints:

$1 <= n <= 10^2$

$-10^9 <= e11, e12 \ldots nxm$ elements $<= 10^9$

All rows and columns are sorted in increasing order

Steps:

d = 43

11	12	13	14
21	22	23	24
31	32	33	34
41 (starting point)	42	(43)	44

1. Take the starting point as a[a.length-1][j]
   => i = a.length-1, j = 0
2. Now, we will apply the Binary search
3. If a[i][j] > data : i--; else if a[i][j] < data : j++
   else : print the data indices
4. continue till i >= 0 && j < a.length

Dry Run

i = a.length-1 = 3, j = 0
i >= 0 && j < a.length ✓
a[i][j] = 41 < 43, so
j++
i = 3 >= 0, j = 1 < a.length
a[i][j] = 42 < 43 so
j++
i = 0, j = 2 < a.length
a[i][j] = 43 = 43 (3, 2)

Program:

```
int i = a.length-1;
int j = 0;
while (i >= 0 && j < a.length) {
    if (a[i][j] == data) {
        Println(i);
        Println(j);
    }
    return;
    else if (a[i][j] > data)
        i--;
    else j++;
}
Print ("Not found");
```

String, StringBuilder And ArrayList

```
int n=sc.nextInt();
String str=sc.nextLine();
```

nextLine() – It reads till found enterline

next() – till spaces it read

Input	output	Input	Output
19	19	19 Hello	19
Hello			Hello

↓ To avoid this

```
int n= Integer.parseInt(sc.nextLine());
String str=sc.nextLine();
```

```
String str= "hello";   str.length() -5
                       str+='e' - helloe
                       str.charAt(3) -l
```

Setting the characters is not allowed

Sub-string : continous part of string

```
String str="hellobye";
String s1= str.substring(1, 6);
```

Inclusive, Exclusive - ellob

```
str.substring(3,3)- Blank
str.substring(4) – Reads from 4th characters [obye]
```

contains: checks whether a string contains a sequence of characters. println(str.contains("be")) → true

split: splits the string into string array based on delimiter. String[] =Str.split(" ");

toCharArray converts string into char Array

char c[] = str.toCharArray();

replace : Replaces character group of characters with another characters and gives a new string.

str = str.replace("el.", "lo");

startsWith: Returns a string which contains starts with index character.

1. Print All Pallindromic Substrings

Input:	Output:	Constraints
abcc	a b c cc c	1 <= length of string <= 500

Steps:

1. abcc

All the substrings are

a, b, c, c

ab, bc, cc

abc, bcc

abcc

Among all only the coloured ones are pallindromes.

2. Fix the outer loop that iterates through the string with i

3. Take another loop with j = i+1 and generate the substring

4. Check whether the substrings are pallindrome or not by comparing the characters from first to last.

DryRun:

str: abcc

i=0 < 4

j = i+1 = 1 <= 3

sub = str.substring(0,1) : a → Return true print a

Palindcheck comparing characters first and last, if they are not equal false. otherwise true indicating they are pallindromes

j = 2 <= 4

sub = (0,2) : ab → Return false

j = 3 <= 4

sub = (0,3) : abc → Returns false

j = 4 <= 4

sub = (0,4) : abcc → Returns false

i = 1 < 4

j = i+1 = (1,2) : b → Return true

j = 3 <= 4 = (1,3) : bc → Returns false

j = 4 <= 4 (1,4) : bcc → Returns false

i = 2 < 4

j = i+1 = 3 = (2,3) : c → Returns true

j = 3+1 = 4 = (2,4) : cc → Return true

j = 5 <= 4 X

i = 3 < 4

j = 4 <= 4 = (3,4) : c → Returns true

Program;

```
for(int i=0; i<str.length(); i++){
    for(int j=i+1; j<=str.length(); j++){
        ~~if isPalin~~
        String sub = str.substring(i,j);
        if(isPalin(sub))
            println(sub);
    }
}
```

```
Public static boolean isPalin(String str){
    int i=0;
    int j=str.length()-1;
    while(i<=j){
        char a=str.charAt(i);
        char b=str.charAt(j);
        if(a!=b)
            return false;
        else{
            i++;
            j--;
        }
    return true;
}
```

2. String Compression.

Input: wwwwaaadexxxxxx

Output: w4adex
w4a3dex6

Constraints:
1<=length of string<=1000

Steps:

For Compression1:

1. Take an empty string and assign the first character.
2. Now iterate through the string and and add only characters to the new string if consecutive characters are not equal. [ending characters of new string and str array]

For Compression2:

1. Take an empty string and assign the first character
2. Take count=1
3. Check the ending character of new

string and str are equal, then increment the count

else, if count is greater than 1, then assign the count to new string then the character to new string and reassign count=1

4. For the last character count, simply check if count >1, then append count to the string.

Program;

```
public static String compression1(String str1){
    String b=" ";
    b+=b.charAt(0);
for(int i=1; i<str.length(); i++){
        if(b.charAt(b.length()-1)!=str.charAt(i)){
            b=b+str1.charAt(i);
    }
}
return b;
}
Public static String compression2(String str){
    String b=" ";
    b+=str.charAt(0);
    int count=1;
    for(int i=1; i<str.length(); i++){
        if(b.charAt(b.length()-1)==str.charAt(i))
            count++;
        else{
            if(count>1){
                b+=count;
            }
            b=b+str.charAt(i);
            count=1;
        }
    }
    if(count>1)          // For last character
        b+=count;
return b;
```

3. Toggle Case

Input: pepCODinG

Output: PEPcodING

Constraints: 1 <= length of string <= 1000

Steps:

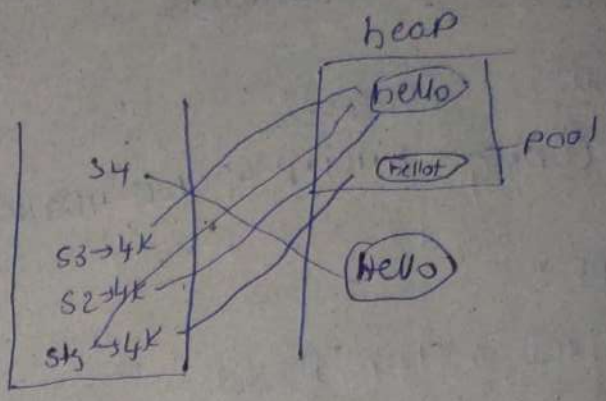
1. Take an Empty String
2. Iterate through the string by converting to charArray.
3. Check if the character is lowerCase, if yes, then convert to UpperCase and append.
4. ~~Check if the~~ Else, append by converting the characters to lowerCase

Program:

```
String ans="";
for(character c: str.toCharArray()){
    if(Character.isUpperCase(c)){
        ans+=Character.toLowerCase(c);
    }
    else
        ans+=Character.toUpperCase(c);
}
```

String Memory;

```
String str = "hello";
String s2 = "hello";
String s3 = "hello";
```



For Allowance,

```
str = str.charAt(4, t)
```

Strings are Immutable.

Purpose of Interning? Space will be saved.

```
str = 'e';
```
» costly

Instances are immutable but references are mutable

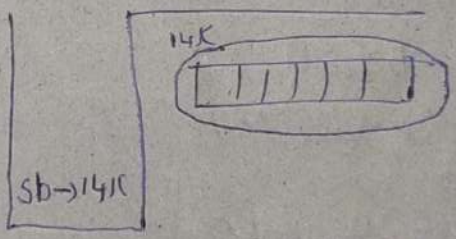
```
String s4 = new String("hello");
```

```
str == s2 ✓
s2 == s4 X
```
== compares on the basis of address

```
s2.equals(s4)
```

String Builder;

```
StringBuilder sb = new StringBuilder();
```
intially 16 size



Methods: deleteCharAt, replace, insert, append

delete(int start, int end)

4 String with Difference of Every Consecutive Characters

Input: PEPCODING

Output: P4E11P-45C1EO-11D37:5N-396

Constraints: 1 <= length of string <= 1000

Steps:

1. Create a String Builder
2. Go through the string and first and consecutive characters
3. Subtract the characters, append the first character and difference.
4. Append the last character to the String Builder
5. Convert the StringBuilder into string and return the string

Program:

```
StringBuilder sb = new StringBuilder();
for(int i=0; i<str.length()-1; i++){
        char a=str.charAt(i);
        char b=str.charAt(i-1);
        int d=b-a;
        sb.append(str.charAt(i));
        sb.append(d);
}
sb.append(str.charAt(str.length()-1));
return sb.toString();
}
```

5. Print All Permutations of a String Iteratively

Input:	Output:	Constraints:
abc	abc bac cab acb bca cba	1 <= length of string <= 15

Steps:

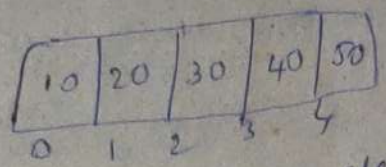
1. No of permutations of a string = Factorial of length of the string.
2. Now iterate from 0-n (i)
3. Take a temp = i and divisor as length
4. Take a StringBuilder sb and append the string to sb
5. Get the remainder r = temp % div and ~~assign~~ print the respective position of sb
6. After printing, delete the character, modify temp = quotient and decrement div
7. This process continues till div >= 1
8. After each permutation Enter into a new line

Program:

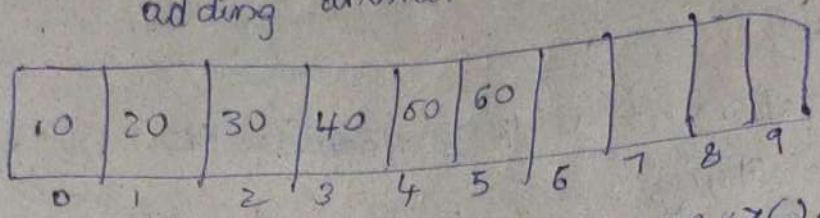
```
int n = fact(str.length());
For(int i=0; i<n; i++) {
        int temp = i;
        int div = str.length();
StringBuilder sb = new StringBuilder(str);
    while (div >= 1) {
        int r = temp % div;
        int q = temp / div;
        Print(sb.charAt(r));
        sb.deleteCharAt(r);
        temp = q;
        div--;
    } println();
}
```

Array List:

Dynamically growing Array



adding another element




```
ArrayList<Integer> al=new ArrayList<>();
    al.add(10);
    al.add(20);
    al.add(30);
    al.add(40);
    al.add(50);
for(int i=0;i<al.size();i++)
    print(al.get(i))
```

1
10
20
30
40
50

al.set(2,1000) → changing value

al.add(2,1000);

10 20 1000 30 40 50

6. Remove Primes

Input	Output:	Constraints
4	(12, 15)	1<=N<=10000
3 12 13 15		

Steps:

1. Go through Each Element of Array List if it is prime, remove the element from Array List or increment the i value

Program:

```
for (int i=0; i<al.size();) {
    int k = al.get(i);
    if (isPrime(k)
        al.remove(i);
    else
        i++;
public static boolean isPrime(int n) {
    for (int j=2; j*j<=n; j++) {
        if (n%j==0)
            return false;
    }
    return true;
}
```

7. Count Palindromic Substrings:

Input:	Output	Constraints
abccbc	9	9

Steps:

1. Every single character in a string is Palindromic substring.
2. Next: a | b c c bc , left = i-1, right = i+1
   For Every left >= 0 & right < a.length, str.charAt(left) == str.charAt(Right) increment the count, we will get the odd palindromic substrings
3. Next: a | b c c b c , left = i, right = i+1
   silly we get even palindromic substrings

Dry Run:

Str = abccbc

ans = Str.length = 6 ← For single character

i = 0, i < 6 ✓

left = 0 - 1 = -1 odd case
right = i + 1 = 1
left >= 0 ✗

left = i = 0
right = i + 1 = 1
left >= 0, right < str.length(),
str.charAt(left) != str.charAt(right)
a != b ✗ even case

i = 1, i < 5
left = 1 - 1 = 0
right = 2
left >= 0 ∧ right < str.length()
a != c ✗ odd case

~~i = 2, i < 5~~

left = i = 1
right = 2 even case
b != c

i = 2, i < 5
left = 1
right = 3 odd case
b != c ✗

left = 2
right = 3 even case
c == c ✓
ans = 7,
left -- = 1
right = 4

b==b

ans=8,

left=0

right=5

right<str.length

5<6 ✓

a!=c ✗

i=3<6,

left=2 odd case

right=4

c!=b ✗

left=3 even case

right=4

c!=c ✗

i=4<6, left=3 odd case

right=5

c==c ✓

ans=9

~~even case~~ left=2

right=6

right<6 ✗

~~i=5<6~~, left=4 even case

right=6<6 ✗

i=5<6, left=4 odd case

right=6<6 ✗

left=5

right=6<6 ✗ even case

i=6<6 ✗

Program;

```
int ans = str.length();
for(int i=0; i<str.length(); i++) {
    int left = i-1;
    int right = i+1;
while(left >= 0 && right < str.length() &&
str.charAt(left) == str.charAt(right)){
    ans++;
    left--;
    right++;
}
left = i;
right = i+1;
while(left >= 0 && right < str.length()
  && str.charAt(left) == str.charAt(right))
{
ans++;
left--;
right++;
}
}
println(ans);
```

Recursion:

Sum of 1st n natural numbers

$$\sum_{i=1}^{n} f(n) = \frac{n(n+1)}{2}$$

-Principle of Mathematical Induction

1. Prove that $N=1$, $\sum_{i=1}^{1} = 1$, $= \frac{1 \times (1+1)}{2} = 1$
2. Assumption this formula is true, for $n=k$

$$\sum_{i=1}^{k} = \frac{k(k+1)}{2}$$

3. Prove for $k+1$

$$= \underbrace{1+2+3+\dots+k}_{\text{from 1 2}} + k+1$$

$$= \frac{k(k+1)}{2} + (k+1)$$

$$= (k+1)\left[\frac{k}{2}+1\right]$$

$$= \frac{(k+1)(k+2k)}{2}$$

$$= \frac{(k+1)^2((k+1)+1)}{2}$$ ← Same as $\frac{n(n+1)}{2}$

1. Print Decreasing

Input	Output	Constraints:
5	5	$1 <= n <= 1000$
	4	
	3	
	2	
	1	

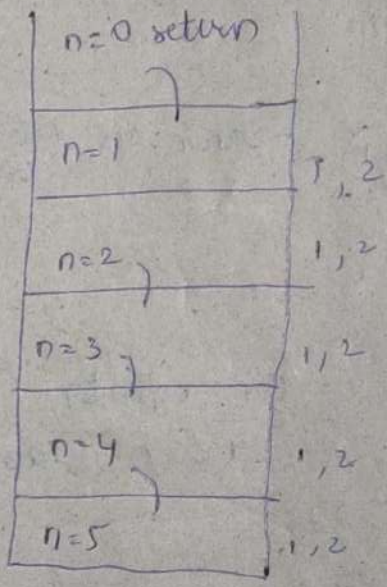
Steps:

1. If n == 0 return
2. Else print n and call the function with n-1

Program:

```
Public static void printDecreasing(int n){
        if n==0
            return;
        else{
            println(n);  -①
            printDecreasing(n-1); -②
}
```

Dry Run:



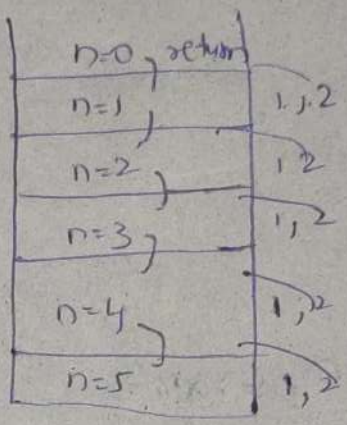
2. Print Increasing

Input	Output	Constraints
5	1 2 3 4 5	$1 <= n <= 1000$

Steps:

1. If n==0, return else call f(n-1) recursively and return the n values

Dry Run:



Program:

```
if n==0
    return 0;
else {
    printIncreasing(n-1);
    Println(n);
}
```

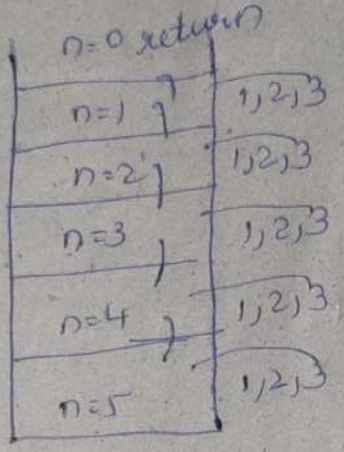
3. Print Increasing Decreasing

Input	output	Constraints
5	5 4 3 2 1 1 2 3 4 5	$1 <= n <= 1000$

Program;

```
if n==0
    return;
else
    printIn(n);  -①
    pd;(n-1);    -②
    printIn(n);  -③
```

Dry Run;



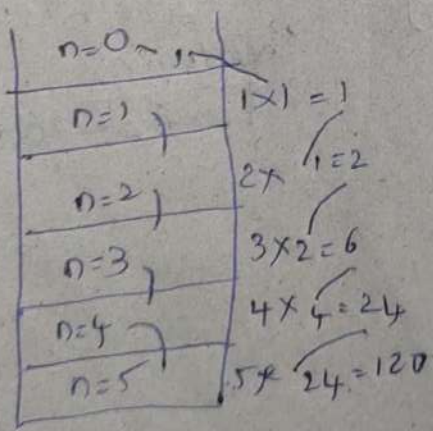
4. Print factorial

Input	Output	Constraints
5	120	$0 <= n <= 10$

Program;

```
n==0
    return 1;
else
return n* factorial(n-1);
```

DryRun;



5. Power-linear

Input	Output	Constraints
2	32	$1 <= x <= 10$
5		$0 <= n <= 9$

Steps; If n==0 return 1
else return $x * Fun(x, n-1)$.

Program;
```
if n==0
    return 1;
else
    return x * power(x, n-1);
```

Dry Run:

x=2, n=0	
x=2, n=1	2^1 = 2
x=2, n=2	2 x 2 = 4
x=2, n=3	2 x 4 = 8
x=2, n=4	2 x 8 = 16
x=2, n=5	2 x 16 = 32

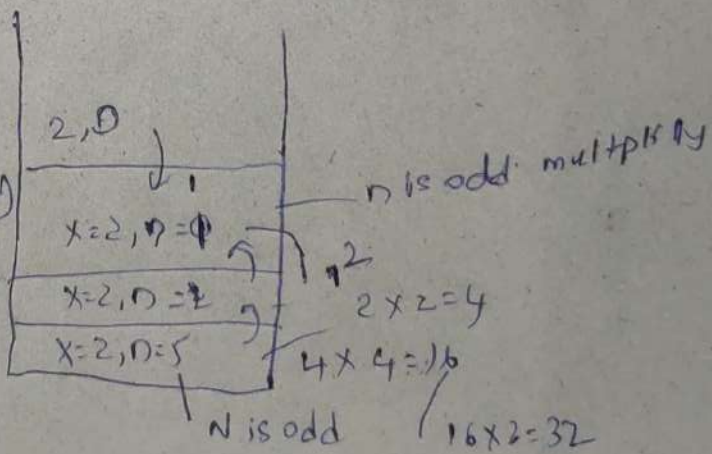
6. Power-logarithmic

Input	Output	Constraints
2	32	$1 <= x <= 50$
5		$0 <= n <= 9$

Steps:
1. For logarithmic, we will have temp = pow(x, n/2)
2. ans = temp * temp
3. If n is odd multiply with x

Dry Run:

x	n	temp	ans
2	5	4	16 x 2 (n is odd) = 32
2	2	2	4
2	1	1	1 x 2 (n is odd)
2	0		



Program:

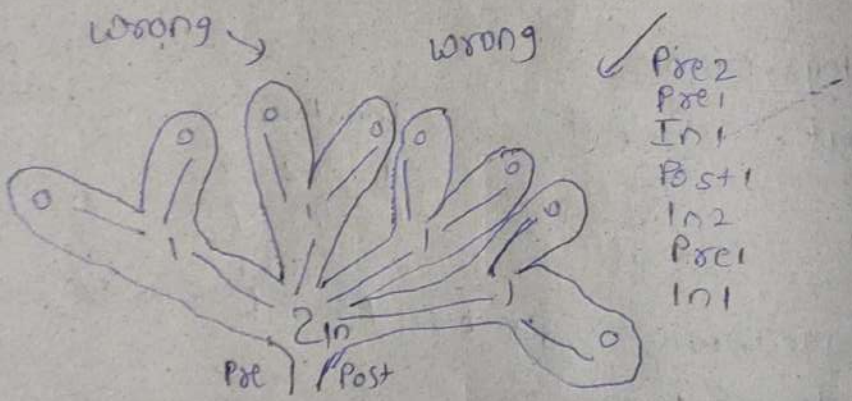
```
if n == 0
    return 1;
else {
int temp = power(x, n/2);
int ans = temp * temp;
if (n%2 == 1)
        ans = ans * x;
return ans;
```

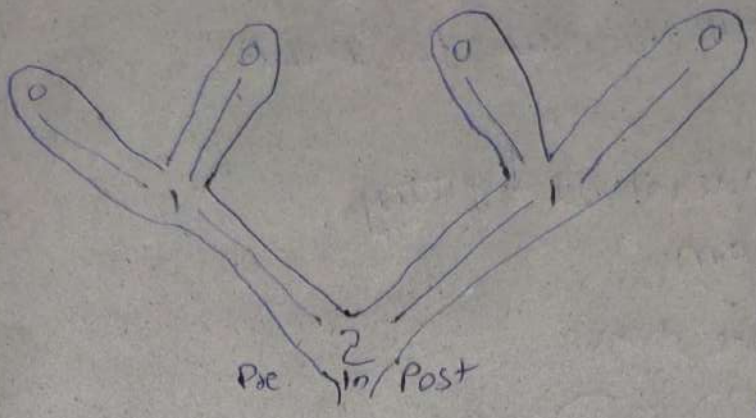
7. Print ZigZag

Input	Output	constraints
3	3211 2111 2321 1211 23	1 ≤ n ≤ 10

Steps:

Understanding By Euler Diagram





Pre 2
Pre 1
In 1
Post 1
In 2
Pre 1
In 1
Post 1
Post 2

Program;

```
if n==0
    return;
else {
    print(n+" ");
    pzz(n-1);
    print(n+" ");
    pzz(n-1);
    print(n+" ");
}
```

8 Tower of Hanoi

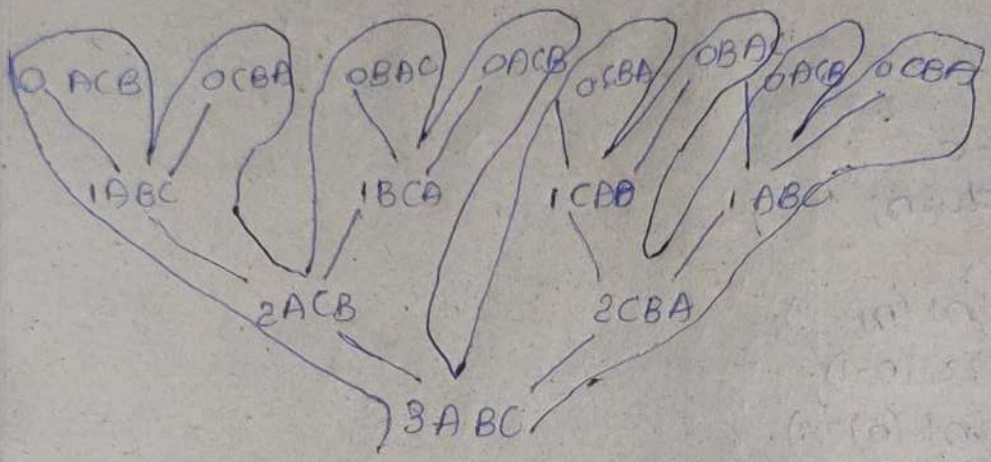
Input; 3
10
11
12

Output:
1 [10 → 11]
2 [10 → 12]
1 [11 → 12]
3 [10 → 11]
1 [12 → 10]
2 [12 → 11]
1 [10 → 11]

Constraints:
$0 <= n <= 9$
$10 <= n1, n2, n3 <= 10^9$
$n1 != n2 != n3$

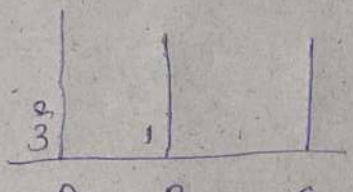
Steps:

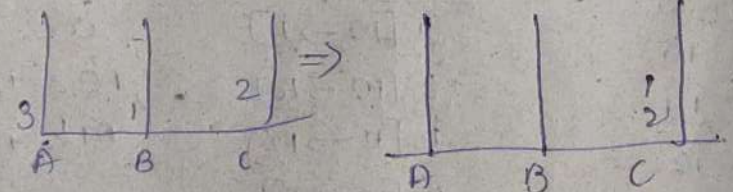
1. We will assume that 2 disks are following all the rules for 3 disks problem.
2. We will understand by using Euler Diagram.
3. There are three disks

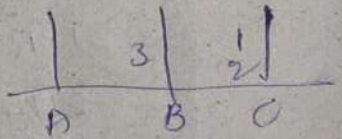


Point only the 1s instructions

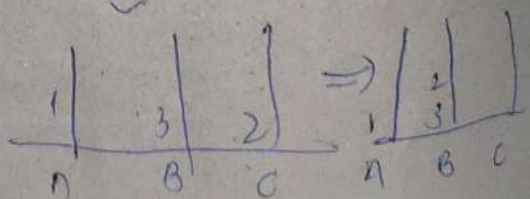
1ABC
2ACB
1BCA
3AB
1CA
2CB
1AB

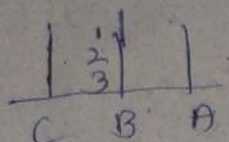






!!





Program:

```
int n= sc.nextInt();
int t1d=sc.nextInt();
int t2d=sc.nextInt();
int t3d=sc.nextInt();
toh(n,t1d,t2d,t3d);

static void toh(int n, int t1d, t2d, t3d){
    if n==0
        return 0;
    else{
        toh(n-1, t1d, t3d, t2d);
        println(n+"("+t1id+"->"+t2id+")");
        toh(n-1, t3id, t2id, t1id);
    }
```