{Binary Search}

Page No: .....
Date: .....

→ Brute → Linear Search.

① Lower Bound.

↳ Given → Array
↳ x.

→ A Sorted Array.
→ Need to find Smallest index such that Arr[idx] >= x.

T.C ② {10, 20, 30, 30, 40, 50, 60, 70} (indices 0 1 2 3 4 5 6 7)

x = 40 ↳ ④.
x = 30 ↳ ②
x = 35 ↳ ④
x = 80 ↳ ⑧
x = 5 ↳ ⓪.

No ~~ele~~ equal or greater ele ~~Return~~ Return Length of Arr

Optimal → Binary Search.

T.C ①
Arr = {1, 2, 3, 4, 5, 6, 7} (indices 0 1 2 3 4 5 6)
x = 1 → Ans ⓪

```
lo=0, hi=n-1 ; Ans = n;
while (lo <= hi) {
    mid = (lo+hi)/2 ;
    if (Arr[mid] >= x){
        Ans = mid
        hi = mid-1;
    }
    else {
        lo = mid+1;
    }
}
```

② Upper Bound → Same As Lower Bound

↳ Just Condition is Arr[idx] > x. Not equal too.

T.C Arr = {10, 20, 30, 30, 40, 50, 60, 70} (indices 0 1 2 3 4 5 6 7)

x = 30 → lb = ② , ub = ④
x = 35 → lb = ④ , ub = ④
x = 5 → lb = ⓪ , ub = ⓪
x = 80 → lb = ⑧ , ub = ⑧.

⇒ Same Code & Just ~~Con~~ Change the Condition.

③ Search Insert Position → Given → Array (Sorted) → Target.

You Need to Find Correct index where you can insert Target.

T.C ① {1, 3, 5, 6} (indices 0 1 2 3)
t = 5 → Ans ②

② {1, 3, 5, 6} (indices 0 1 2 3)
t = 2. → Ans ①

If target is already in Array so return that index. Else return the index ~~of~~ just greater ele then target.

Brute → Linear Search.
↳ if (Arr[i] >= t) return ①

Optimal → Lower Bound.
↳ Same Code. And Same Logic.

④ Floor And Ceil in Sorted Array → Given → Array (Sorted) → x.

Floor ↓ Largest No. in Array <= x

Ceil → Smallest No. in Array >= x. (Lower Bound)

* Brute force
↳ Linear Search.
↳ Store Based on Condition.

* Optimal :-
↳ Logic Binary Search
↳ 1st find Ceil
↳ then Floor.
} use two diff methods

```
findFloor(Arr, n, x){
    ans = -1
    while(lo <= hi){
        if(Arr[mid] <= x){
            ans = Arr[mid];
            lo = mid + 1;
        }
        else hi = mid - 1;
    }
}
```

```
findCeil(Arr, n, x){
    ans = -1;
    while(lo <= hi){
        if(Arr[mid] >= x){
            ans = Arr[mid];
            hi = mid - 1;
        }
        else lo = mid + 1;
    }
}
```

T.C ① {3, 4, 4, 7, 8, 10} x = 5
Ans → {4, 7}
4 → floor, 7 → Ceil

② {3, 4, 4, 7, 8, 10} x = 8
Ans → {8, 8}
8 → floor, 8 → Ceil

---

⑤ First And Last Occurrence → Given → Array (sorted), target
→ Need to find 1st & Last Occurrence of target.

T.C ① Arr = {5, 7, 7, 8, 8, 10} t = 8
Ans → {3, 4}

② Arr = {5, 7, 7, 8, 8, 10} t = 6
Ans → {-1, -1}

Brute Force → Linear Search
→ Update → 1st Occure Variable if(ele == t) Only one time
→ Last Occure Variable Every time

Optimal → Lower Bound
→ Use LB for firstOccurrence.
→ Then using another variable iterate from 1st Occurrence to find Last Occurrence.

Better → LB & UB.
→ 1st Occurrence → LowerBound
→ Last → UpperBound - 1.
→ If target is Not in Array in this case you need to check before return.

```
firstOcc = findLB(Arr, x, n);
if(firstOcc == -1 || Arr[firstOcc] != x){
    return {-1, -1};
}
else {
    i = firstOcc;
    while(Arr[i] == Arr[firstOcc]){
        i++;
    }
    return {firstOcc, i-1};
}
```

---

⑥ Search In Rotated Sorted Array - I.
→ Given → Rotated Sorted Array (Distinct values), k
→ Need to find the index of k (target) ele. if Not return (-1).

T.C ① {4, 5, 6, 7, 0, 1, 2} k = 0
Ans → 4

k = 3
Ans → -1

Optimal → Compare mid ele with hi or lo & based on that move the pointer.

if(Arr[mid] <= Arr[hi]).
→ You are in Left Sorted Array

Brute Force → Linear Search.

if(Arr[lo] <= Arr[mid]).
→ You are in Right Sorted Arr.

```
if(Arr[mid] == t)  return mid;
if(Arr[mid] >= Arr[lo]) {  // You Are in Right Sorted Array.
    if(t > Arr[mid] && t <= Arr[hi])
        lo = mid + 1;
    else
        hi = mid - 1.
}
else { } → Check Same for Left Sorted Array.
```

Page No: .....
Date: .....

---

(7) Search In Rotated Sorted Array — II.

→ Same As Part-I.
→ But In this Array will have duplicate Values.
→ Need to Return True or False.

T.C # Arr = {7, 8, 1, 2, 3, 3, 3, 4, 5, 6}. k = 3.

Ans → True.

Optimal: → Similar to Part-I.

→ Just Add one Condition.

```
if(Arr[mid] == Arr[hi] && Arr[mid] == Arr[lo]) {
    lo++;
    hi--;
    continue;
}
```

→ Because we Already checked that ele At mid is not equal to target.

Arr = {3, 1, 2, 3, 3, 3, 3} lo → 0, hi → 6, mid → 3, pc = 1.

Trim Array to this.

→ So, ele At lo, mid, hi is same. we Not needed ele at lo & hi.

Brute Force → Linear Search.

---

(8) Find Minimum In Rotated Sorted Array.

Given → Rotated Sorted Arr
→ Need to return min ele in Array.

T.C {4, 5, 6, 7, 0, 1, 2}

Ans → 0

Brute Force → Linear Search

Optimal

Arr = {4, 5, 6, 7, 0, 1, 2} lo, mid (min), hi

Left part is sorted so take min from it And Eleminate.

→ Same Concept {0, 1, 2} lo, mid, hi

Logic:- Find the Sorted Part or Right. Take the minimum & Eliminate That Part.

→ Right Part is Sorted. So mid ele will be min.

```
if(Arr[mid] <= Arr[hi]) {
    minM = min(minM, Arr[mid]);
    hi = mid - 1;
}
else {  // Left Part is sorted.
    minM = min(minM, Arr[lo]);
    lo = mid + 1;
}
```

min will be At lo.

(9). Find Out How Many Times the Array is Rotated → Distinct Ele. Page No: Date:

↳ Given → Array (Sorted & ~~Right~~ Right Rotated).

↳ Need to Return How many time's it is Rotated

T.C Arr = {4, 5, 6, 7, 0, 1, 2, 3}

Ans (4) It is Rotated 4 times.

Brute Force → Linear Search

↳ Find the min^m ele in Array & its index.

↳ That index is the Answer.

#Optimal :- → Use Binary Search to find the min^m ele And its index.

↳ Use Logic of Previous Question.

```
if( Arr[lo] <= Arr[hi]){
    if (Arr[lo] < Ans){
        Ans = Arr[lo];
        idx = lo;
        break;
    }
}
```

Just Optimize with this condition.

Logic :- → Initially min^m ele is At 0th position

↳ In Every rotation the No. from Right Comes to ~~z~~ front

↳ So the No. of times min^m ele is shifted is the No. of Rotation.

*. {4, 5, 1, 2, 3, 4} (lo, hi)

Why to check mid. → Whole ~~Arr~~ is Sorted part.

---

(10). Single Element in Sorted Array. → Given → Sorted Array.

↳ Every No. Appears twice.

↳ Except One. Need to find & Return.

T.C :- {1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 6}

Ans → (4)

#Optimal. → BS.

⇒ Observation.

0	1	2	3	4	5	6	7	8	9	10
1	1	2	2	3	3	4	5	5	6	6
	even	odd	Even	odd			odd	even	odd	even
		Even	odd							

Left Half | Right Half.

#Brute Force (1) → Check every ele.

↳ Ele ~~should~~ should equal to its ~~next~~ or previous

↳ if Not Return that Ele

(2) → Use Xor.

↳ Xor of same No. will give you zero.

↳ So duplicate No.'s will be cancle out.

if (even, odd) → I am in Left ~~half~~ of element.

if (odd, even) → I am in Right half of element. → So element will be in opposite half.

lo = 1 hi = n-1. → for (0th And (n-1)th) handle them seperately.

```
while( lo <= hi){
    if ( Arr[mid] != Arr[mid+1] && Arr[mid] != Arr[mid-1] ) return Arr[mid];

    if ( (mid%2 == 1 && Arr[mid] == Arr[mid-1]) || (mid%2 == 0 && Arr[mid] == Arr[mid+1]) )
        ~~# this mid + 1~~ lo = mid + 1;
    else
        hi = mid - 1;
}
```

(11). Find Square Root of A Number. → Give → (n).
↳ Return its Square Root.

Page No: .....
Date: .....

T.C → n = 36 (Ans → 6). n = 28 (Ans → 5) → integer value. √28 => 5.292.

Optimal → Binary search

```
lo = 0  hi = n.
while(lo <= hi){
    val = mid * mid;
    if(val <= n)  → Ans = mid.
        lo = mid+1;
    else
        hi = mid-1;
}
```

Brute force → Linear search

```
for(i = 1 → n){
    val = i * i;
    if(val <= n)
        ans = i;
    else
        Break;
}
```

(12). Find Nth root Of A Number. → Given → (N) → Need to find A No. which can multiple N times & gets equal to M.
↳ M.

T.C (1) N = 3, M = 27. (Ans → 3). 1×1×1 = 1, 2×2×2 = 8, 3×3×3 = (27) ✓

(2) N = 4, M = 69. (Ans → -1). 1×1×1×1 = 1, 2×2×2×2 = 16, 3×3×3×3 = (81) > (69).

Brute force → Linear search

```
for(i = 1 → M){
    val = Pow(i, N);
    if(val == M)
        return i;
    else if(val > M)
        break;
}
```

Optimal → Binary search

```
lo = 1  hi = M;
while(lo <= hi){        // Use Optimized pow function.
    midN = Pow(mid, N);
    if(midN == M)   else if(midN < M)
        return mid;     lo = mid+1;   else
                        hi = mid-1;
}
```

(13). Find the Smallest Divisor. → Given → Array, limit. {if(n > limit) return -1}

→ You need to take an Divisor which will divide every ele in Array. After that the sum of array should be less then limit. If it is less then that Divisor might be an ~~Ans~~ Answer But you need to find Smallest.

T.C Arr = {1, 2, 5, 9}, limit = 6. (Ans → 5).

D = 4: 1/4 + 2/4 + 5/4 + 9/4 ⇐ 1 + 1 + 2 + 3 = (7) > 6. So (4) is Not an Answer.
(you will take ceil value)

D = 5 → Ans: 1/5 + 2/5 + 5/5 + 9/5 → (1) + (1) + (1) + (2) = (5) ≤ 6 ✓

You can directly return the ans because it will be pointing to Answer. At the end lo will be pointing to Answer.

Brute Force → Linear Search

↳ At max the Answer will be max^m element in Array
↳ So do LS from 1 to max^m element.

Optimal → Binary Search from 1 to max ele in Array.

```
while(lo <= hi){
    Sum = (nums, mid);
    if(sum <= limit){
        Ans = mid;
        hi = mid-1;
    }
    else lo = mid+1;
```

Page No: .... Date: ....

(14) Koko Eating Bananas

Koko Eating Bananas → Given → Array of piles {Every pile has some No. of Bananas} ; h (hours)

- Koko need to eat All the bananas in h (hours).
- In every 1 hr he can eat some bananas (k). we need to minimize.
- if pile has less bananas than (k). then Koko will eat All of them. But Not able to move to next pile he need to wait till next hour starts.

At max it will eat [max^m No. in Array] No. of Bananas in hr.

TC Arr = {7, 15, 6, 3} h = 8.

Ans (5): (5) bananas in (1) hr. → (2)+(3)+(2)+1 = (8) hr

Brute Force → Linear Search
- from (1) to (max^m ele.)
- if No. of hours is Less then store the Ans.

Optimal → Binary Search
- Range is same As Brute.

```
lo = 1, hi = max(Arr);
while (lo <= hi) {
    TotalHr = CalcTotalHr(Arr, mid);
    if (TotalHr <= h) {
        Ans = mid;
        hi = mid - 1;
    }
    else {
        lo = mid + 1
    }
}
```

mid → [mid No. of Bananas per hr.]

---

(15) Minimum Days to Make M Bouquets

→ Need to Return min^m No. of Days to Create m Bouquets

Given:
- Array → Every index is A flower And it is Storing No. of Day it will take to bloom
- m [No. of Bouquets you need to create]
- k [No. of flowers needed to make 1 Bouque]

we have (8) flowers in total. But Not Bloomed

TC Arr = {7, 7, 7, 7, 13, 11, 12, 7} (indices 0–7), m = 2, k = 3

- m = 2 → Need to Create (2) Bouques.
- k = 3 → Every Bouque must contain (3) flowers

This flower will be blommed After (7) day. → (12)

You can Only take the Adjacent Bloomed flowers to create one Bouquete.

```
boolean possible(Arr[], day, m, k) {
    count = 0, NoOfBou = 0;
    for (i = 0 → n) {
        if (Arr[i] <= day) {
            count++;
        }
        else {
            noOfBou += count/k;
            count = 0;
        }
    }
    noOfBou += count/k;
    return (noOfBou >= m);
}
```

Brute force → Linear Search
- from min^m ele to max^m ele in Array.
- If you are Able to Create Bouquets then Return that (i).

Optimal → Same Like Linear Search
- Just do Binary Search on that Range.

```
while() {
    if (possible(Arr, mid, m, k)) {
        ans = mid;
        hi = mid - 1;
    }
    else lo = mid + 1;
}
```

(16) **Aggressive Cows.**

Given → Array → Coordinates / position of Stalls.
Given → K (No. of Aggressive Cows)

We Need to Assign stalls to Cows such that distance b/w any two cows is min^m. And you need to return maximum of all possible combination.

T.C Arr = {0, 3, 4, 7, 10, 9} Cows = 4.

0 3 4 7 9 10.

C1 ↔(3) C2 ↔(1) C3 ↔(3) C4. ⇒ (1) ⇒ Ans ⇒ (3)

C1 ↔(4) C2 ↔(5) C3 ↔(1) C4. ⇒ (1)

C1 ↔(4) C2 ↔(3) C3 ↔(3) C4. ⇒ (3)

Return max^m of this.

Brute Force → Linear Search
1. Sort the Array.
limit = Arr[n-1] - Arr[0]
(max^m possible distance b/w two Cow.)
Loop from 1 to limit.
Try to maintain At least (i) distance b/w two Consicute Cows.

```
boolean canWePlace(Arr[], dist, cows){
    cntCows = 1; last = Arr[0];
    for(i = 1 → n){
        if(Arr[i] - last >= dist){
            cntCow++;
            last = nums[i];
        }
        if(cntCows >= cows)
            return true.
    }
    return false.
}
```

Optimal → Binary Search

```
lo = 1, hi = limit;
while( ){
    if(canWePlace(Arr, mid, k))
        Ans = mid;
        lo = mid + 1;
    else
        hi = mid - 1;
}
```

Ans will be (hi)

(17) **Book Allocation Problem** ** Hard.

Given → Array. Every (i) is Book & it stores No. of Pages in that Book.
Given → m [No. of Students]

You need to distribute All the Books to All students.
Each Student At least get One Book. [You need to Allocate Books in Continuous manner]

Max^m No. of Pages Assign to any particular student is the min^m possible

T.C Arr = {25, 46, 28, 49, 24}. Students = (4).

(Ans. (71))

S1	S2	S3	S4	max^m
→ (25)	(46)	(28)	(49+24)	⇒ (73)
→ (25)	(46)	(28+49) max	(24)	⇒ (77)
→ (25)	(46+28) max	(49)	(24)	⇒ (74)
→ (25) (46)	(28)	(49)	(24)	⇒ (71) min^m

max

Ans

Brute Force → Linear Search

~~min One St for get is max pages~~

St from max^m ele in Array to sum of All element.

Then count the No. of students, if it matches return the Answer.

Optimal → Do Binary Search on same Range.

Page No: .....
Date: .....

```
int CountStudents(Arr[], pages) {
    Stud = 1, pagestudent = 0;
    for(int i = 0 → n) {
        if(pagestudent + Arr[i] <= pages) {
            pagestudent += Arr[i];
        }
        else {
            Stud++;
            pagestudent = Arr[i];
        }
    }
    return Student stud;
}
```

```
lo = max(Arr);
hi = Sum(Arr);
while(lo <= hi) {
    stud = CountStudent(Arr, mid);
    if(stud <= m)
        lo = mid + 1;
    else {
        Ans = mid;
        hi = mid - 1;
    }
}
```

---

(18). Find Peak Element. → Given → Array

→ Peak Ele → Arr[i-1] < Arr[i] > Arr[i+1]; ← Peak element

→ Need to return its index.

→ If multiple are there. Return Any One.

T.C. Arr = {0:1, 1:2, 2:3, 3:4, 4:5, 5:2, 6:1}

→ Ans = 4

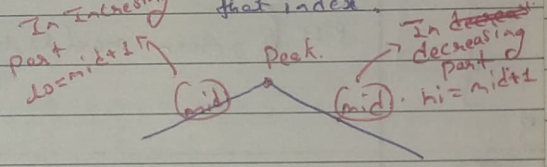
Arr = {-1, 0:1, 1:2, 2:3, 3:4, 4:5, 5:-1}

→ Ans ⇒ 5

Brute Force → Linear Search

→ Check the Condition.

→ if true Return that index.



lo = 1, hi = n-2.

```
while(lo <= hi) {
    if(Arr[mid] > Arr[mid-1] && Arr[mid] > Arr[mid+1]) {
        return mid;
    }
    if(Arr[mid] > Arr[mid-1]) → In Left part
        lo = mid + 1;
    else
        hi = mid - 1; → In Right Part.
}
```

Optimal → Binary Search

→ Check 0th & n-1th index seperately.

---

(19) Median Of 2 Sorted Arrays. → Given → Array 1, Array 2. → Sorted.

→ Need to return the Median.

T.C. Arr1 = {2, 4, 6}.

Arr2 = {1, 3, 5}.

Ans → 3.5, (3+4)/2.

Brute Force

→ Create New Array of m+n size

→ merge two sorted Arrays in new one

→ Return the Median.

Better → use Merge two Sorted Array Logic.

→ use 2 extra variables ele1, ele2.

→ Also maintain Count.

→ if count = (m+n)/2 → You find ele1 Store it

→ if count = (m+n)/2 - 1 → You find ele2 Store it.

if(m+n) → Even → Return (ele1 + ele2)/2.

→ odd → return ele1

Optimal.

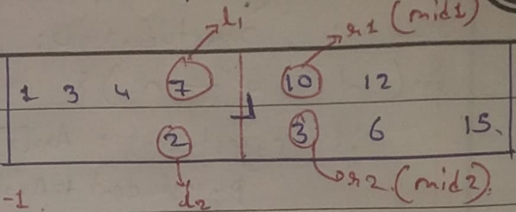
1, 2, 3, 3, 4 | 6, 7, 10, 12, 15.

Page No: ....

Date: ....

Arr1 = {1, 3, 4, 7, 10, 12} → n1

Arr2 = {2, 3, 6, 15} → n2.



(4) ✗ l1 > r2 ↳ hi = mid-1.

left = (n1 + n2 + 1) / 2.

lo = 0, hi = n1.

while( ) {

mid2 = left - mid1;

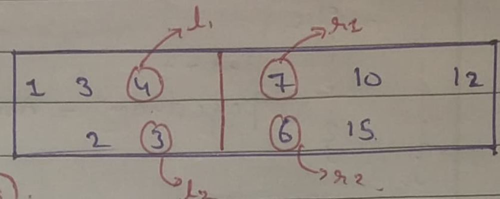
l1 = (mid1 > 0) ?
Arr1[mid1-1] : Int_Min;

r1 = (mid1 < n1) ?
Arr1[mid1] : Int_Max;

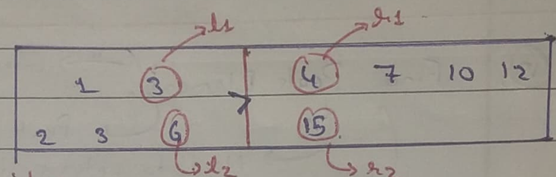
l2 = (mid2 > 0) ?
Arr2[mid2-1] : Int_Min;

r2 = (mid2 < n2) ?
Arr2[mid2] : Int_Max;

(3) ✓. l1 <= r2, l2 <= r1 ↳ (Ans).



(2) ✗. l2 > r1 ↳ lo = mid+1.




```
if (l1 <= r2 && l2 <= r1) {
    if ((n1 + n2) % 2 == 1)
        return Max(l1, l2);
    else { return (max(l1, l2) + mid(r1, r2)) / 2.0;
}
else if (l1 > r2) {
    hi = mid1 - 1;
}
else {
    lo = mid1 + 1;
}
}.
```

(20). Minimize Max Distance to Gas Station

↳ Given → Array [Positions of Gas Stations] → sorted.
→ k [New Gas Stations]

↳ You need to place New Gas Stations.
→ In A way that Minimize max^m distance b/w two Gas Stations.
→ You can not move Original Gas Stations. You can only put new one's b/w them, so that distance will minimize.

T.C Arr = {1, 2, 3, 4, 5}
k = 4 (Ans) 0.5

⇒ {1, 1.25, 1.5, 1.75, 2, 3, 4, 4.5, 5} — (1) max (between 2 and 3)

⇒ {1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5} — (0.5) max

k = 5
Arr = {1, 13, 17, 23} n
howMany = {0, 0, 0} n-1

* Denotes How many New Gas Stations are there b/w 1st & 2nd Gas Stations.

Brute Force
→ You will Run loop k times.
→ And every time find the max^m difference.
→ Because you will try to minimize it.
→ So, you will put new Gas Station there.

Page No: .....
Date: .....

```
for(i = 1 ⟶ k) {
    maxSection = -1; maxInd = -1;
    for(j = 0 ⟶ n) {
        diff = Arr[j+1] - Arr[i];
        sectionLeng = diff / howMany[j] + 1;
        if (sectionLeng > maxSection) {
            maxSection = sectionLeng;
            maxInd = i;
        }
    }
    howMany[maxInd]++;
}
```

Better.

You can optimize this section Little Bit.

→ Because to find max^n section. Every you are ittrating in entire Array.

→ You Heap (Max).

→ It will give you max^n section in O(1).

⇒ Optimal is Real Question, so skipping Optimal.

---

(21) K^th Element Of Two Sorted Arrays.

Given → Arr1, Arr2 → Sorted

→ k.

→ Need to Return k^th element.

T.C

Arr1 = {2, 3, 6, 7, 9}

Arr2 = {1, 4, 8, 10} k=5.

→ 6.

Optimal → median of two Sorted Arrays logic.

k=5. k → ele

mid1 → l1, r1 → n-k ← ele

2 3 6 | 7 9

1 4 8 10.

l2, r2 ← mid2

```
if (l1 <= r2 && l2 <= r1)
    return Math.max(l1, l2);
else if (l1 > r2)
    hi = mid1 - 1;
else lo = mid1 + 1;
```

Better → use Merge Two Sorted Array logic.

```
while(i<m && j<n) {
    if(Arr[i] <= Arr[j]){
        count++;
        if(count==k)
            return Arr[i];
        i++;
    }else{
        count++;
        if(count==k){
            return Arr[j];
        }
        j++;
    }
}
```

---

(22) Split Array - Largest Sum. → Given → Array, k.

→ You need to split the Array into k Non-empty subarrays.

→ In A way that Largest Sum of any Subarray is minimized.

I.C

Arr = {10, 20, 30, 40}. K=2.

{10} {20, 30, 40} → 90 (max)

{10, 20} {30, 40} → 70 (max)

{10, 20, 30} {40} → 60 (max) ← min^n.

Ans → 60 min^n

Painter's Partition → Exactly same Question.

Logic → Exactly same to Allocate Books.

So These three Questions are exactly has same logic.

(23) Find row with Max^n 1's.

Page No: .....
Date: .....

Given → Grid → of n x n.
Consist only 0's, 1's.
All the rows are sorted.

Need to return row which has max^n No. of 1's.

T.C

	0	1	2	3	4 (j)
0	0	0	1	1	1 (i)
1	0	0	0	0	0
2	0	1	1	1	1
3	0	0	0	0	0
4	0	1	1	1	1

Brute Force
- Count of every row.
- And store max^n one.
- Return it.

Optimal.

```
i=0 , j=n-1
if(Arr[i][j] == 1){
    ind = i;
    j--;
}
else
    i++;
```

(24) Search in A 2D Matrix

Given → Matrix → of m x n.
Row wise sorted
1st ele of every row is greater than last ele of previous row.
target.
Need to Return True or Flase. Target exist or Not.

Optimal.

T.C

	0	1	2	3
0	3	4	7	9
1	12	13	16	18
2	20	21	23	29

t = 23.

Think like this. flatten A 2D into 1D.

{ 3, 4, 7, 9, 12, 13, 16, 18, 20, 21, 23, 29 }
0 1 2 3 4 5 6 7 8 9 10 11

lo → 0
hi → (m x n) - 1
mid = (5)
(5)/m → row.
(5)%m → Col.

Brute Force
- Itrate in entire matrix.

Better → Check 1st & Last ele of every Row.
- If target is b/w if. Do Binary Search on that row.

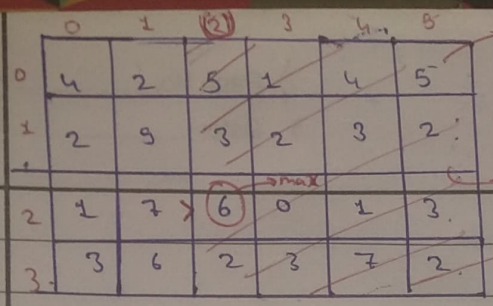
```
if(Arr[row][col] == t)
    return true;
else if(Arr[row][col] < t)
    lo = mid+1;
else hi = mid-1;
```

(25) Find Peak Element. Given → Matrix (m x n)
You Need to return Peak Element.
In Matrix No two Adjacent Cells are equal.
Peak Element → which is strictly greater than All of its Adjacents top, bottom, left, Right.
Assume Entire matrix is Surrounded by (-1).

Optimal → Logic.
- Do Binary Search on Col's.
- Then on that mid Column find the max^n ele.
- That max ele will be strictly greater then top & Bottom. Then check for its left & Right & eleminate the

Brute Force
- Itrate in entire matrix And find the max ele. And return its coordinates.

Page No.



Page No: .....
Date: .....

find max ele in that col.

Top & Bottom is solved.

Left 6 Right → Check for Left & right.

max ele.

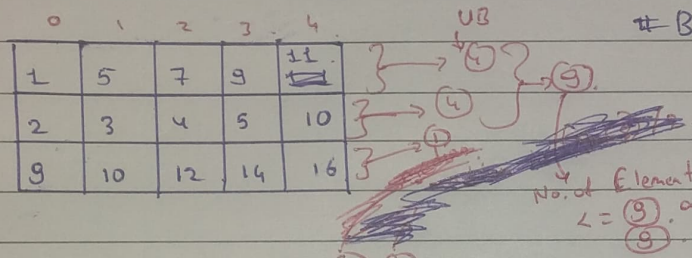
```
if (left > max ele)
    hi = mid - 1.
else if (right > max ele)
    lo = mid + 1.
else
    return element.
```

* Left or Right may Not exist, so Consider it as -1 or Int_Min.

---

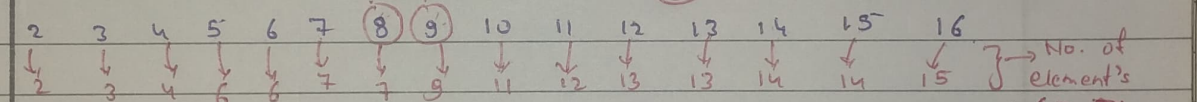
(26) Matrix Median. → Given → Matrix → which is row wise sorted.
→ Need to find the Median.

T.C



Brute Force.

→ Create New List And All the ele's in List
→ Sort the List & return the median based on Length.



Optimal → Do Binary search from min^n ele to max^n ele in matrix.

→ for every mid. Calculate the No. of less then & equal to elements in matrix
→ If it is <= required of lo = mid + 1.
→ else hi = mid - 1.

lo = min^n hi = max^n (Min^n will be at 0th col & max^n will at n-1th col)

```
req = (n*n)/2 ;
while (lo <= hi){
    SmallEqual = CountSmallEqual(Matrix, mid);
    if(SmallEqual <= req)
        lo = mid+1 ;
    else
        hi = mid-1;
}
return lo;
```

```
CountSmallEqual(matrix, mid){
    int Count = 0;
    for (i=0 → matrix.length){
        Count += UpperBound(matrix[i], mid);
    }
    return Count;
}
```