Search 2: Binary Search Problems

Use Cases

Binary Search has numerous applications:

- → Searching in database.
- → Finding element in sorted array.
- → Finding an insertion point for a new element in a sorted array.
- → Implementing features like autocomplete in search engines.

Question :- Search in a rotated sorted array of distinct element.

Eg :- 2, 4, 8, 10, 15

Rotations → K = 1 → 15 2 4 8 10

K = 2 → 10 15 2 4 8

Task is to find element 'x' in this rotated array.

Edge case → K = 5 → 2 4 8 5 10

Q How you will figure out whether an Array is rotated or Not ?

↳ compare first & last Value.

→ if (A[0] > A[N-1]) { rotated }
else Not rotated.

→ If not rotated

↳ Last class first problem.

→ Remaining Case → Array is rotated.

Ex:- 4, 5, 8, 10, 1, 2, 3

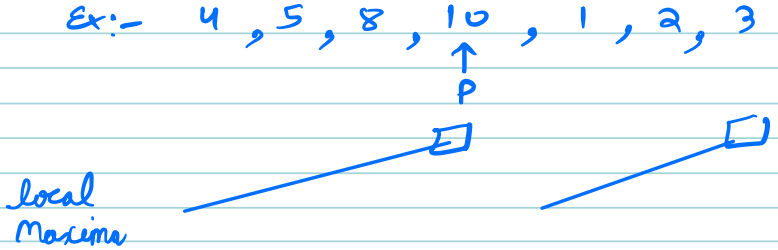
Constraints :- If the index of largest element is given in following way, then we can apply search

↳ from (0, index)

↳ from (index+1, N-1)

⇒ Twist ! :- If the index of largest element is not given.

Ex:- 4, 5, 8, 10, 1, 2, 3



Approach:- Apply Binary Search (BS) to find the local Maxima & then Apply BS on Left & Right Side.

↳ 3 Binary Search

Twist 2 :- Do it in 1 Binary Search

Ex:- 4, 5, 8, 10, 1, 2, 3 (4, 5, 8, 10 → Part 1; 1, 2, 3 → Part 2)

Part 1 > Part 2

Q → How will you figure out that to which part "X" belongs

↳ (x < 0th element) {

| Part 2

} else {

| Part 1

}

IDEA:- Get Mid, find in which part our middle is & in which our target is, if both are in different parts, will move mid towards the target else apply normal binary search

DRY RUN

K = 20

0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	1	2	3	4	5	6	7	8	9

L	R	Mid	Mid Area	Target Area
0	11	5	2	1 → Goto Left
0	4	2	1	1 → Goto Left (Apply BS)
0	1	0	1	1 → Goto Right (Apply BS)
1	1	1	Break	

Ex

K = 60

0	1	2	3	4	5	6
70	80	90	100	40	50	60

L	R	Mid	Mid Area	Target Area	
0	6	3	1	2	→ Goto Right
4	6	5	2	2	→ Goto Right.
6	6	→	found		

→ Diagramatic

① 

↳ goto right

② 

↳ Goto Left

③ 

↳ Apply Normal Binary Search.

PSEUDO CODE

```
L = 0 , R = N-1
while ( L <= R) {
    m = L + ((R-L)/2);
    if ( A[m] == target) {
        return m
    }
    if ( target < A[0]){ // target → part2
        if ( A[m] >= A[0]){ // mid → part1
            L = m+1
        } else { // mid → part2
            if ( A[m] < target){
                L = m+1
            } else {
                R = m-1
            }
        }
    } else { // target → Part 1
        if ( A[m] < A[0]){ // mid → part2
            R = m-1
        } else { // mid → part1
            if ( A[m] < target){
                L = m+1
            } else {
                R = m-1
            }
        }
    }
}
```

return -1;

TC →	O(log N)
SC →	O(1)

Quiz 1 :- In the problem of searching for a target element in a rotated sorted array, what advantages does Binary Search offer over Linear Search?

↳ Binary Search divides the search space into half at each step.

Question :- Given the No. , find Square root of N

Note :- Return int (Ans) or floor (sqrt(N))

Ex: sqrt(25) → 5

sqrt(20) → 4

sqrt(10) → 3

Brute force

$\sqrt{20} \rightarrow 4$

1, 2, 3, 4

```
i = 1;
ans = 0;
while (i*i <= N){
    ans = i;      → this can be potential ans.
    i++;
}
return ans;
```

SC —	O(1)
TC —	O(√N)

$1 \to N$ $1 \to \sqrt{N}$

↳ OPTIMIZATION

↳ Binary Search

⇒ Over here the search space is not given directly. **Right ??**

1. Target → floor(sqrt(N))
2. Search space → (1 → N)
3. Conditions to discard

Case I :- mid * mid == N
return mid.

Case II :- mid * mid < N → Potential Ans
ans = mid
L = mid + 1 **Go right**

Case III :- mid * mid > N
R = mid - 1 **Goto Left**

DRY RUN **N = 50**

L	R	mid	
1	50	25	→ 25 * 25 > 50 → Goto Left
1	24	12	→ 12 * 12 > 50 → Goto Left
1	11	6	→ 6 * 6 < 50 → Goto Right, Ans = 6
7	11	9	→ 9 * 9 > 50 → Goto Left
7	8	7	→ 7 * 7 < 50 → Goto Right, Ans = 7

8 8 8 → 8 * 8 > 50 → Goto Left

8 7 break

PSEUDO CODE

↳ H.W.

TC → O(logN)

SC → O(1)

Break :- 10:22 – 10:32

Question :- A^{th} Magical Number

→ Multiples of 3 in range [1 100]

→ $\frac{100}{3} = 33$

→ Multiples of 4 in range [1 100]

→ $\frac{100}{4} = 25$

→ Multiples of 6 in range [1 100]

→ $\frac{100}{6} = 16$

→ Multiples of 4 or 6 in range [1 100]

→ $\frac{100}{4} + \frac{100}{6} - \underbrace{\frac{100}{LCM(4,6),12}}_{\text{Removing common elements}}$

Ex:- 4 → 4 8 12 16 20 24 28...

6 → 6 12 18 24 30.....

Generalize

Multiple of A & B in the range of 1 to X = $\frac{X}{A} + \frac{X}{B} - \frac{X}{LCM(A,B)}$

Problem Statement :- A, B & C is given find A^{th} Magical No.

↳ A no. is said to be magical if it is divisible by B or C

Ex :-

B	C	A
2	3	8

Magical No :- 2 3 4 6 8 9 10 12 ....

↑ 8^{th} Magical No.

Ex :-

B	C	A
4	6	5

Magical No. :- 4 6 8 12 16

↑ 5^{th} Magical No.

Brute force

↳ move from 1 to A^{th} Magical No.

PSEUDO CODE

```
count = 0;
for ( i=1 ; i<=min(B,C)*A ; i++) {
    if (i%B == 0 || i%C == 0) {
        count++
    }
    if (count == A) {
        return i
    }
}
```

↳ OPTIMIZATION

Q Till what No. I can make sure that we will get A^{th} Magical No. ?

→ Min(B, C) * A

Now, ① Search Space

↳ [min(B,C) , Min(B, C)*A]

② Target → A^{th} Magical No.

③ Condition to discard

B	C	A
4	6	10

Q:- Is 36 my 10^{th} Magical No. ?

$$\frac{36}{4}^{9} + \frac{36}{6}^{6} - \frac{36}{12}^{3} = 12$$

↳ with this we can find multiple of B or C less than 36

So Goto Left

1, 2, ... 8 ..... 36 .....

X

Q:- is 24 my 10th Magical No. ?

$$\frac{24}{4}^{6} + \frac{24}{6}^{4} - \frac{24}{12}^{2} = 8$$

So, Goto Right

DRY RUN

B	C	A
5	7	4

Min(B,C) L	Min(B,C)*A R	Mid	
5	20	12	$\frac{12}{5} + \frac{12}{7} - \frac{12}{35} = 3$ → Goto Right.
13	20	16	$\frac{16}{5} + \frac{16}{7} - \frac{16}{35} = 5$ Goto left
13	15	14	$\frac{14}{5} + \frac{14}{7} - \frac{14}{35} = 4$

Q Does it mean 14 is my 4th Magical No.?

Ex:- B=5, C=7, A=3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count→ 0	0	0	0	1	1	2	2	2	3	3	3	3	4	4

Observation :- At multiple of B or C count is incresed by 1

Conclusion :- If formula results int A, then store & try to check Left

PSEUDO CODE

```
Ath_Magical_No(A, B, C){
    L = min(B, C)
    R = A * min(B, C);
    LCM = lcm(B, C);
    while(L <= R)
        m = L + (R-L)/2
        Count = m/B + m/C - m/LCM
        if(Count < A){
            L = m+1;
        }else if(Count > A){
            R = m-1
        }else{
            ans = m
            R = m-1;
        }
    }
    return ans;
}
```

TC → O(log(min(B, C) * A))

SC → O(1)

Question :- Median of 2 Sorted Arrays

Median :- Centre element of sorted array

if (length %2 == 0), then the average of 2 centre elements.

Ex :- A = { 4 10 15 }

B = { 1 5 }

↳ { 1 4 5 10 15 } → Median

Ex :- A = { 4 10 15 }

B = { 1 5 12 }

↳ { 1 4 5 10 12 15 } → $\frac{5+10}{2} = 7.5$

Brute force

↳ Merge two Sorted Arrays & return Median.

TC → O(N+M)

SC → O(N+M)

5 | 5

↳ OPTIMIZATION

↳ Binary Search.

Ex:- A[] → [1 3 4 7 10 12]

B[] → [2 3 6 15]

↳ 1 3 4 | 7 10 12

2 3 | 6 15

↳ Task is to find the boundary because once found the ans will be :-

① Find max of Left

i.e max(4,3)

② Find min of Right

i.e. min(7,6)

③ If length is even.

then, $\frac{\max(4,3) + \min(7,6)}{2}$

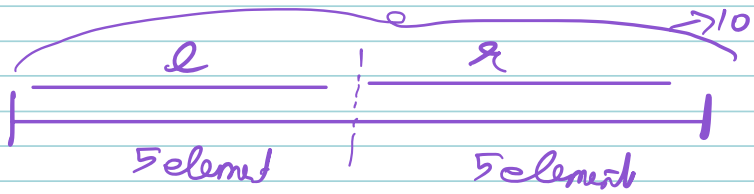
If length is odd

then, max(4,3)

Given we take one extra element in left part.

=> Lets work on the idea of partition

IDEA



$l <= r$

① Target → Median

② Search Space → 2 sorted arrays.

③ Condition to discard

Ex:- A[] → [1 3 4 7 10 12]

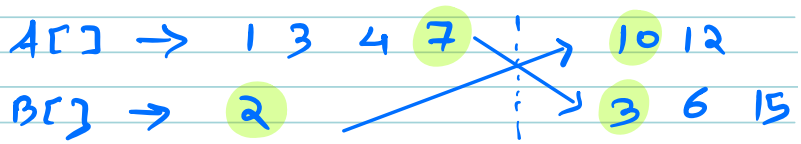
B[] → [2 3 6 15]

Case I:- we choose 4 element for Left of A

A[] = 1 3 4 7 | 10 12

Q:- How many elements I need choose from B for left part

↳ (A.Length + B.Length + 1)/2 — No. of elements in Left of A



↳ Not a Correct partition

Goto Left

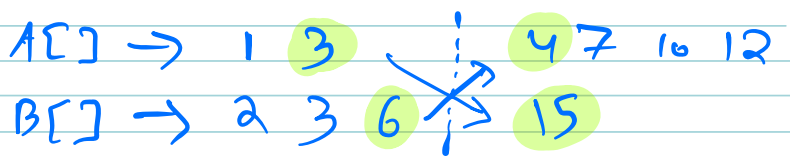
Q:- We should select more than 4 or less than 4 in A?

=> Less elements in A.

∵ max of A on Left > Min of B on Right

→ Goto Left.

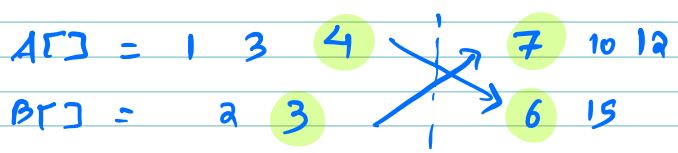
Case II, we choose 2 from A for Left part



∵ max of B on Left > Min of A on Right

↳ Goto Right

Case 3, we choose 3 elements from A for left part



↳ Got Correct partition, return median.

PSEUDO CODE

```
findMedian ( A[] , B[] ) {
    if ( A.Length > B.Length ) {
        // Swap (A, B) to store shorter in A
    }

    TL = A.Length + B.Length

    L = 0 ;
    R = A.Length ;

    while ( L <= R ) {

        partition1 = (L + R) / 2
        partition2 = (L + R + 1) / 2 - partition1

        maxLeftA = -∞
        maxLeftB = -∞

        minRightA = ∞
        minRightB = ∞

        if ( partition1 > 0 ) {
            maxLeftA = A[ partition1 - 1 ];
        }

        if ( partition2 > 0 ) {
            maxLeftB = B[ partition2 - 1 ];
        }

        if ( partition1 != A.Length ) {
            minRightA = A[ partition1 ]
        }

        if ( partition2 != B.Length ) {
            minRightB = B[ partition2 ];
        }
```

```
if ( maxLeftA <= minRightB && maxLeftB <= minRightA){
    // Return Mid based on length
} else if ( maxLeftA > minRightB)
    R = partition1 - 1
} else {
    L = partition + 1
}
```

TC → O(log(min(A.length, B.length)))

SC → O(1)

quiz 3:- you have an array of integers with an odd nos. of elements which of the following statement is true about finding the median of this array ?

↳ The Median is always the middle element of array.

Observations

- **Sorted Arrays:** Binary Search excels in sorted arrays, capitalizing on their inherent order to quickly locate elements.
- **Search Space:** Identify the range within which the solution exists, which guides setting up the initial search range.
- **Midpoint Element:** The middle element provides insights into the properties of elements in different subranges, aiding decisions in adjusting the search range.
- **Stopping Condition:** Define conditions under which the search should stop, whether an element is found or the search range becomes empty.
- **Divide and Conquer:** Binary Search employs a "divide and conquer" strategy, progressively reducing the problem size.
- **Boundary Handling:** Pay special attention to handling boundary conditions, avoiding index out-of-bounds errors.
- **Precision & Approximations:** Binary Search can yield approximate solutions by adjusting the search criteria.