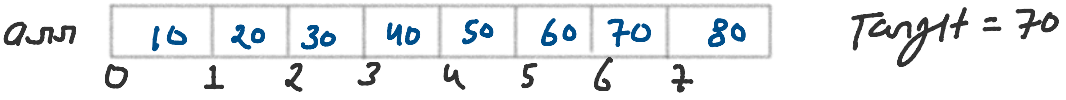
CLASS-11 → 13/09/2023

https://www.linkedin.com/in/manojoffcialmj/

SEARCHING & SORTING
LEVEL-01

1. Linear search

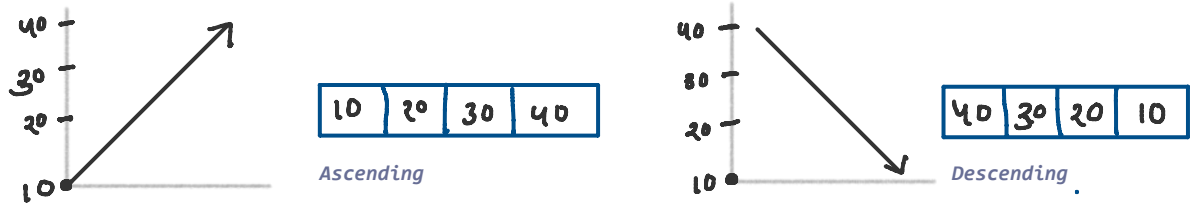


Time Complexity: O(N), where N is length of array.

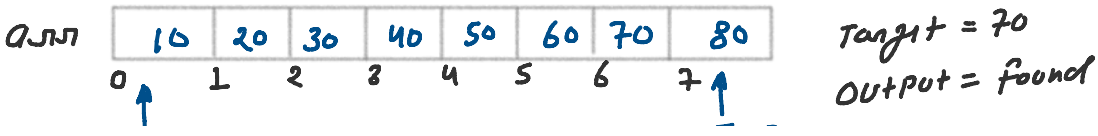
```
for( int i=0; i< arr.size(); i++) {
    if( arr[i] == target) {
        return true;
    }
}
return false;
```

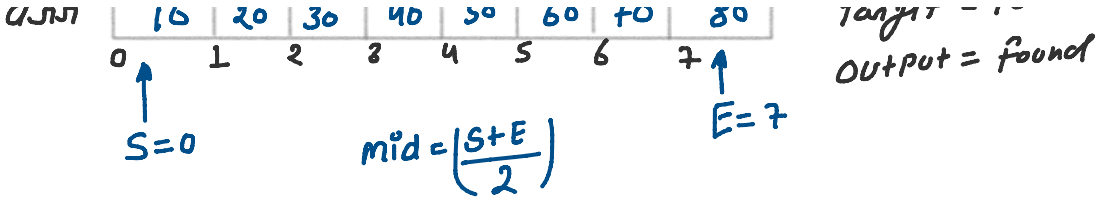
2. Binary search

- *Binary search is always work on **MONOTONIC** array*
- *MONOTONIC -> **Ascending** or **Descending** order of array*



EXAMPLE (1)





Rules

① arr[mid] == target
↳ return true

② arr[mid] < Target
↳ s = mid+1

③ arr[mid] > Target
↳ E = mid-1

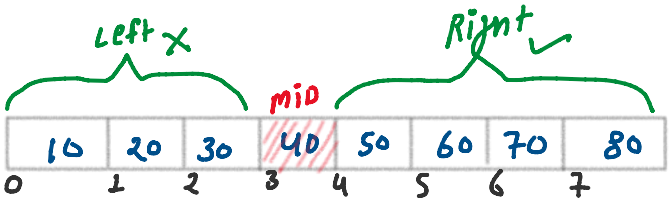
④ Terminating condition
may be → (s < e)
OR
(s <= e)

Note

BEST PRACTICE FOR FINDING MID
↳ Mid = S + (e-s)/2

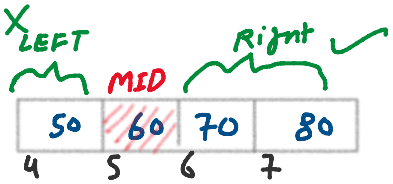
DRY RUN

Situation : 0



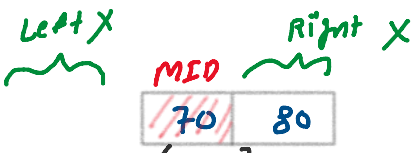
s=0 e=7 mid=3 target=70 Rule ⇒ ② 40 < 70
↳ Right

Situation : 1

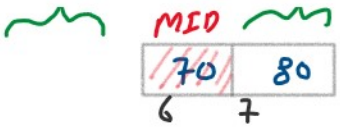


s=4 e=7 mid=5 target=70 Rule ⇒ ② 60 < 70
↳ Right

Situation : 2



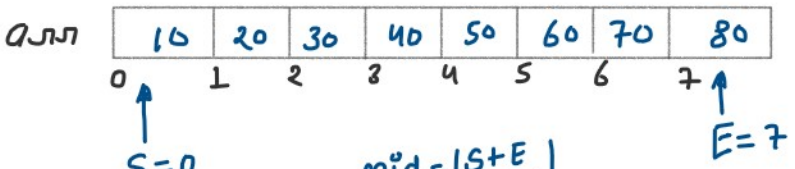
Iteration : 2



s = 6 e = 7 mid = 6 target = 70 Rule ⇒ ① 70 == 70
↳ Return True

Output : found

EXAMPLE ②

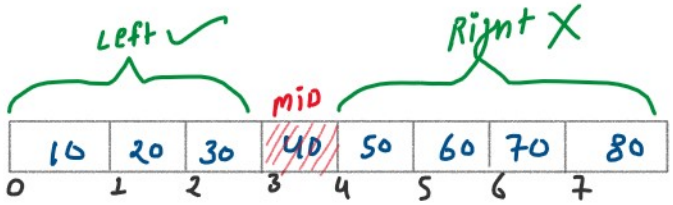


mid = $\left\lfloor \frac{S+E}{2} \right\rfloor$

Target = 20
Output = found

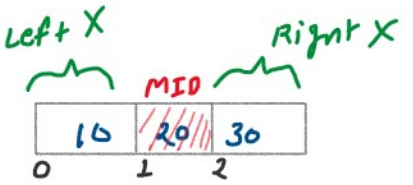
DRY RUN

Iteration : 0



s = 0 e = 7 mid = 3 target = 20 Rule ⇒ ③ 40 > 20
↳ Left

Iteration : 1



s = 0 e = 2 mid = 1 target = 20 Rule ⇒ ① 20 == 20
↳ return True

Output : found

```cpp
// Binary Search
bool binarySearch(vector<int> arr,int target){
    int n=arr.size();
    int start=0;
    int end=n-1;

    // Best practice of finding mid index
    int mid=start+((end-start)/2);

    bool ans=false;

    // Rule 04: Terminating condition to outside of loop
    while(start<=end){
        // Rule 01: When target found the return true
        if(arr[mid]==target){
            ans = true;
            return ans;
        }
        // Rule 02: When target is greater than mid of array value then skip the left part
        else if(arr[mid]<target){
            start=mid+1;
        }
        // Rule 03: When target is less than mid of array value the skip the right part
        else if(arr[mid]>target){
            end=mid-1;
        }
        // Update the mid index (HIGH CHANCE:----->Yaha par me galti kar skta hun)
        mid=start+((end-start)/2);
    }
    // When target not found then return false
    return ans;
}
```

3. Find Time Complexity of Binary Search

size = n

Iteration: 0 — L | R — $n/2^0 \rightarrow n$

Iteration: 1 — L | R — $n/2^1 \rightarrow n/2$

Iteration: 2 — L | R — $n/2^2 \rightarrow n/4$

Iteration: Kth — L ⩽ R — $n/2^{(k-1)} \cong n/2^k$

$$\Rightarrow \frac{n}{2^k} = 1$$

$$\Rightarrow n = 2^k$$

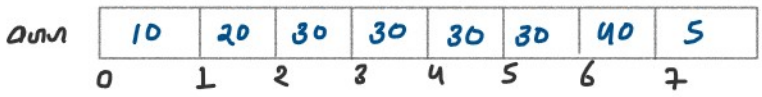
$\Rightarrow \quad n = 2^K$

$\Rightarrow \quad \log_2 n = \log_2(2^K)$

$\Rightarrow \quad \log_2 n = K$ → T.C. $\Rightarrow O(K) \Rightarrow O(\log n)$

RULE : $\log_A A^A = 1$

5. Find first occurrence of a number in sorted array

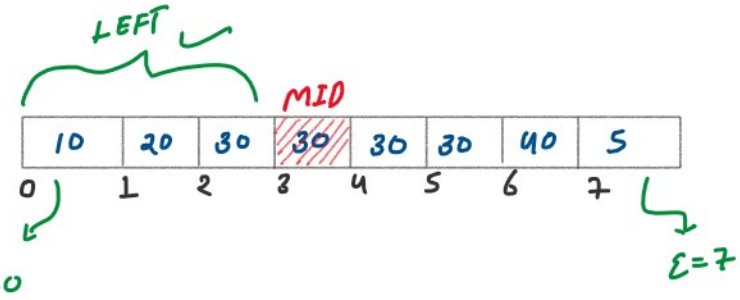


Target = 30

Output = 2

DRY RUN

Situation : 0



$Mid = \frac{0+7}{2} = 3$

Ans = 3

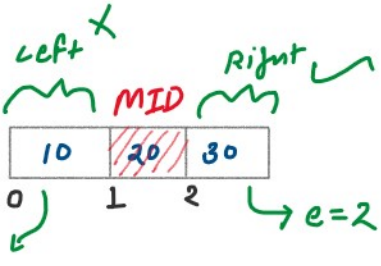
① arr[mid] == Target

→ ans = mid

→ e = mid-1

mid ko Ans main store karlo AND Right part ko SKIP kardo

Situation : 1



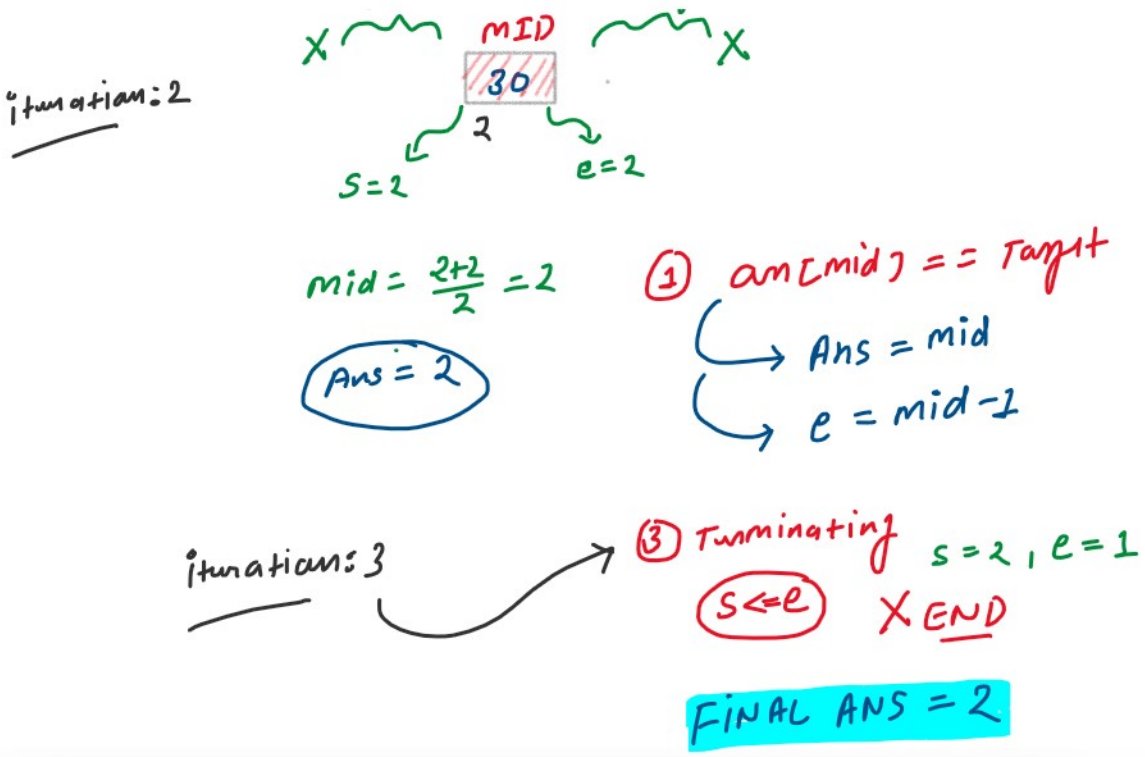
$Mid = \frac{0+2}{2} = 1$

Ans = 3

② arr[mid] < Target

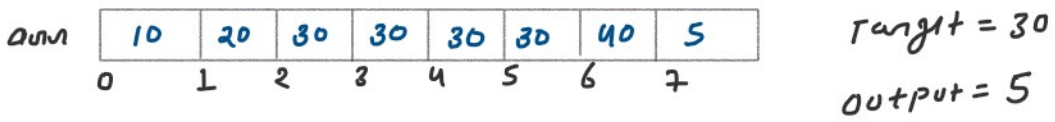
→ s = mid + 1

Situation : 2




```cpp
// First occurrence
int findFirstOccurence(vector<int> arr,int target){
    int n=arr.size();
    int start=0;
    int end=n-1;
    int ans=-1;

    while(start<=end){

        int mid=start+((end-start)/2);

        // Rule 01: When target is equal to mid of array then store possible index in ans and skip the right part
        if(arr[mid]==target){
            ans=mid;
            end=mid-1;
        }
        // Rule 02: When target is greater than mid of array value then skip the left part
        else if(arr[mid]<target){
            start=mid+1;
        }
        // Rule 03: When target is less than mid of array value the skip the right part
        else if(arr[mid]>target){
            end=mid-1;
        }
    }
    // When first occurence of a number is found then return ans=index otherwise ans=-1
    return ans;
}
```

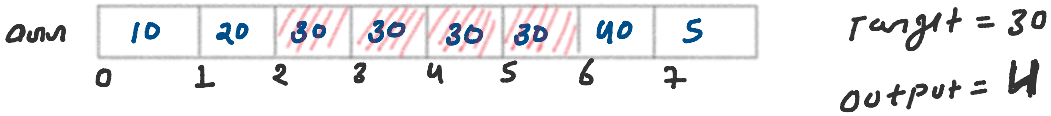
6. Find last occurrence of a number in sorted array



```cpp
// Last occurrence
int findLastOccurence(vector<int> arr,int target){
    int n=arr.size();
    int start=0;
    int end=n-1;
    int ans=-1;

    while(start<=end){

        int mid=start+((end-start)/2);

        // Rule 01: When target is equal to mid of array then store possible index in ans and skip the left part
        if(arr[mid]==target){
            ans=mid;
            start=mid+1;
        }
        // Rule 02: When target is greater than mid of array value then skip the left part
        else if(arr[mid]<target){
            start=mid+1;
        }
        // Rule 03: When target is less than mid of array value the skip the right part
        else if(arr[mid]>target){
            end=mid-1;
        }
    }
    // When first occurence of a number is found then return ans=index otherwise ans=-1
    return ans;
}
```

.

7. Find total occurrence of a number in sorted array



Total OC. = Last OC. − First OC. + 1

= 5 − 2 + 1

= 4 Ans

```cpp
// First occurrence
int findTotalOccurence(vector<int> arr,int target){
    int lastIndex=findLastOccurence(arr,target); // 5
    int firstIndex=findFirstOccurence(arr,target); // 2
    int totalOccurrence=lastIndex-firstIndex+1; // 5-2+1=6-2=4
    return totalOccurrence; // 4
}
/*
EXAMPLE 01:
INPUT: arr[10,20,30,30,30,30,40,50], target=30
OUTPUT: Total occurence of target is 4
*/
```

```
OUTPUT: Total occurence of target is 4
*/
```

8. Find missing element in sorted array (GFG)

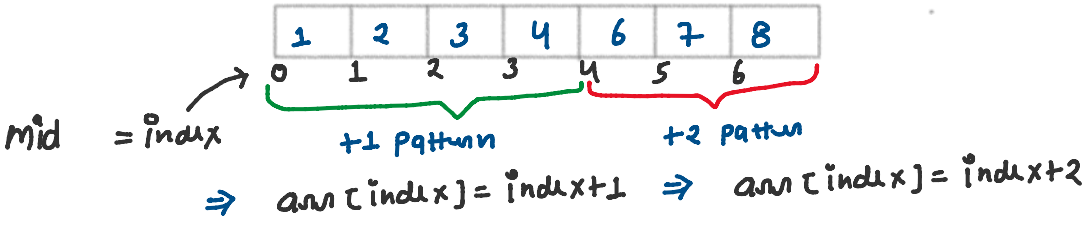
- Given a list of **n-1** integers and these integers are in the range of **1 to n**.
- There are **no duplicates** in list.
- One of the integers is missing in the list.

> **Example 01:**
> Input : arr[] = [1, 2, 3, 4, 6, 7, 8]
> Output : 5

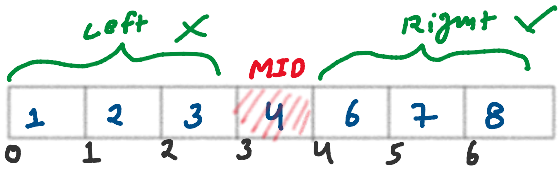
> **Example 02:**
> Input : arr[] = [1, 2, 3, 4, 5, 6, 8, 9]
> Output : 7

DRY RUN

Example 01:



Iteration 0:



s = 0
e = 6
mid = $\frac{0+6}{2}$ = 3

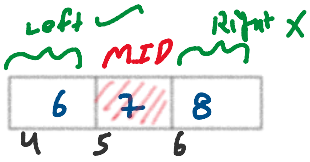
AB pattern check karna hai difference nikal kar

difference = arr[mid] − mid
= 4 − 3
= 1

When (difference = 1) then skip left side

↳ s = mid + 1;

Iteration 1:



s = 4
e = 6
mid = $\frac{4+6}{2}$ = 5

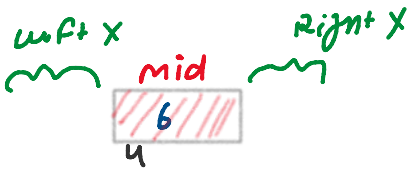
difference = arr[5] − 5
= 7 − 5
= 2

When (difference = 2) then skip right part and store mid in ans

↳ ans = 5 and

↳ ans = 5 and
e = mid - 1

Iteration 2:



s = 4
e = 4
mid = $\frac{4+4}{2}$ = 4

diff[mid] = arr[mid] - mid
= 6 - 4
= 2

↳ (Diff == 2)
ans = 4 and
e = mid - 1

↳ RUK JAO..... KYUNKI
s = 4, e = 3 LEKIN (s <= e)

S>E XEND

↳ AB ANS RETURN KARDO
return (ans+1); 5 missing numbers

DRY RUN

Output = 7

Example 02:

s = 0
e = 7
mid = $\frac{0+7}{2}$ = 3

① diff[mid] = arr[mid] - mid
= 4 - 3
= 1

② (diff[mid] == 1)
↳ s = mid + 1

Iteration 1:

Iteration 1:

$S = 4$

$e = 7$

$mid = \frac{4+7}{2} = 5$

① diffarr = arr[mid] − mid
= 6 − 5
= 1

② (diffarr == 1)
↳ s = mid + 1

Iteration 2:

$S = 6$

$e = 7$

$mid = \frac{6+7}{2} = 6$

① diffarr = arr[mid] − mid
= 8 − 6
= 2

② (diffarr == 2)
↳ Ans = mid
= 6
e = mid − 1

↳ RUK JAO .... KYUNKI
↳ S = 6 , e = 5 ⇒ (S > e) X END

↳ Ans Return Kardo
↳ return (Ans + 1) ;

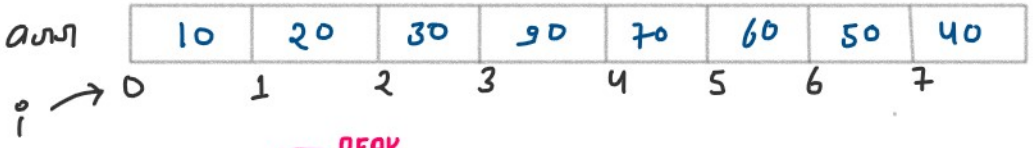
Output ⇒ 7
missing no.

```cpp
// Find missing element
int findMissingElement(int arr[],int size){
    int start=0;
    int end=size-1;
    int mid=start+((end-start)/2);
    int ans=-1;

    while(start<=end){
        // Rule 01: Find the pattern
        int difference=arr[mid]-mid;

        // Rule 02: When difference is equal to 1 then skip the left part
        if(difference==1){
            start=mid+1;
        }
        // Rule 03: When difference is equal to 2 then store mid index in ans and skip the right part
        else if(difference==2){
            ans=mid;
            end=mid-1;
        }
        // Updated mid index
        mid=start+((end-start)/2);
    }
    return ans+1;
}
```

Most Imp

9. Peak element/index in a mountain array (Leetcode-852)

EXAMPLE 1



go=mid+3

Observations

A → arr[i] < arr[i+1]

B → arr[i] > arr[i+1]

PEAK → arr[i] > arr[i+1]
arr[i] > arr[i-1]

Rule:1

arr[mid] < arr[mid+1]
↳ shift to Right start = mid+1

Rule:2

arr[mid] > arr[mid+1]
↳ shift to left / peak point
End = mid → (End = mid-1)

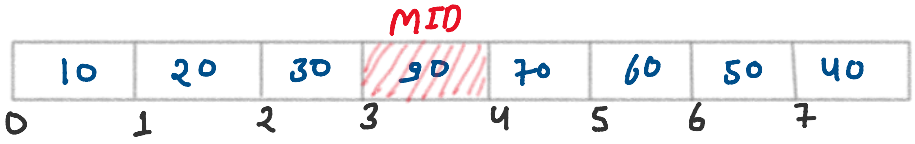
YEH KYUN NAHI KIY? KYUNKI HO SKTA HAI KI YEH KAANE SE PEACK POINT NA MILE.

DRY RUN

MID

DRY RUN

Iteration: 0

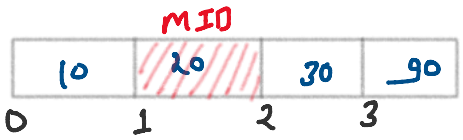


S = 0
E = 7
mid = $\frac{0+7}{2} = 3$

arr[mid] > arr[mid+1]
90 > 70
E = mid

① main B line par HU yaa
② main PEAK par HU

Iteration: 1

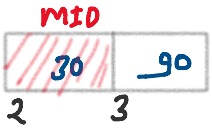


S = 0
E = 3
mid = $\frac{0+3}{2} = 1$

arr[mid] < arr[mid+1]
20 < 30
Start = mid+1

① main A line par HU AND
② Right me PEAK HAI

Iteration: 2



S = 2
E = 3
mid = $\frac{2+3}{2} = 2$

arr[mid] < arr[mid+1]
30 < 90
Start = mid+1

Iteration: 3

S = 3
E = 3
mid = $\frac{3+3}{2} = 3$

AB RUK JAO..... KYUNKI
S = 3, e = 3, S == e
Return karde S KO
return S;

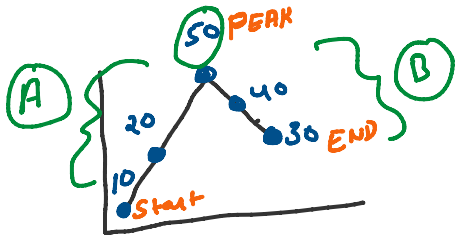
S < e
END

EXAMPLE: 2

EXAMPLE:2

arr

10	20	50	40	30
0	1	2	3	4



Situation:1

MID

10	20	50	40	30
0	1	2	3	4

$S = 0$

$E = 4$

$mid = \frac{0+4}{2} = 2$

arr[mid] > arr[mid+1]

↳ 50 > 40 { ① B clear par ho yaa ② PEAK par hu

End = mid

Situation:0

MID

10	20	50
0	1	2

$S = 0$

$e = 2$

$mid = \frac{0+2}{2} = 1$

arr[mid] < arr[mid+1]

↳ 20 < 50 { ① A line par hu ② Right me PEAK Hai

start = mid+1

Situation:02

50
2

$S = 2$
$e = 2$ } Ruk jao.....(S<e)

↳ Return Kardo S ko

return S

Note — [illegible] Kyunki start always PEAK

Note

S ko hamesa return karenge kyunki start always peak point par exist karta hai at the end me.

```cpp
// Program 06: Peak element/index in a mountain array (Leetcode-852)
class Solution {
public:
    int peakIndexInMountainArray(vector<int>& arr) {
        int n=arr.size();
        int start=0;
        int end=n-1;

        // Termination condition
        while(start<end){
            int mid=start+(end-start)/2;

            if(arr[mid]>arr[mid+1]){
                // Me B line par hu
                // Yaa me PEAK par hu
                end=mid;
            }
            else if(arr[mid]<arr[mid+1]){
                // Me A line par hu
                // And PEAK right me exist karta hai
                start=mid+1;
            }
        }

        return start;

    }
};

/*
Example 1:
Input: arr = [0,1,0]
Output: 1

Example 2:
Input: arr = [0,2,1,0]
Output: 1

Example 3:
Input: arr = [0,10,5,2]
Output: 1

Example 4:
Input : arr[] = [10,20,30,90,70,60,50,40]
Output : 3

Example 5:
Input : arr[] = [10,20,50,40,30]
Output : 2
*/
```