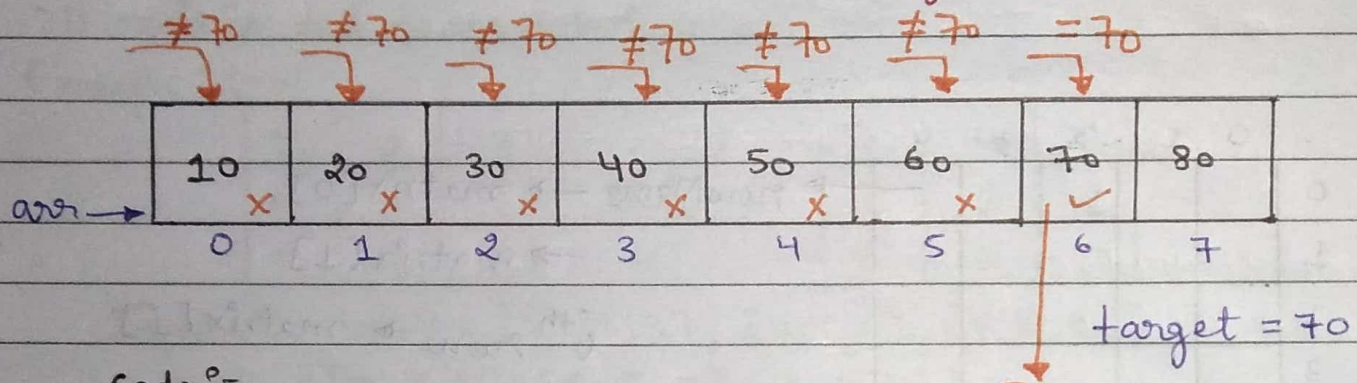
SEARCHING && SORTING
[LEVEL-1]

Date 13/09/23

1. Linear Search :-

It starts searching from one end of a list and goes through each element until the desired target found, otherwise the search continues till the end.

ek ek karke sabko check krdungi



Code :-

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

→ Time Complexity → O(n) → Size of array

2. Binary Search :-

Binary search is only applicable on sorted arrays which can either in ascending order or in descending order.
We can apply it on monotonic functions i.e given content must be in either ascending order / increasing order or in decreasing order.

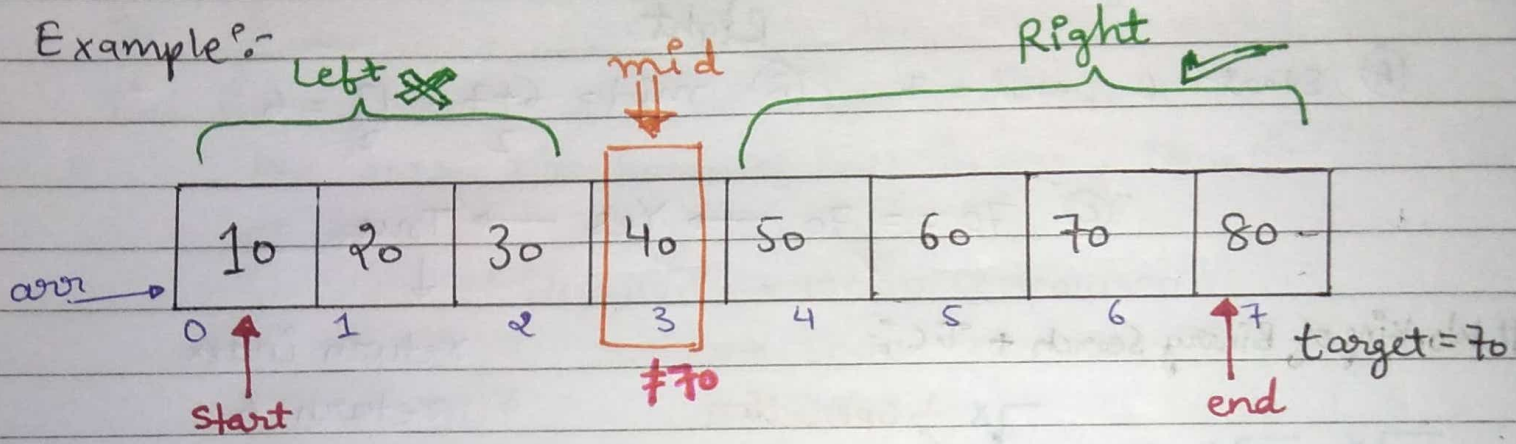
Approach :-

(A) set start and end index with 2 variables
(B) Calculate their mid i.e $mid = \left(\frac{s+e}{2}\right)$
(C) Check the condition and select one part of (Sorted) array and neglect the other.

Date ..../..../.....

BEST PRACTICE — Use start + (end - start) / 2 because mid = (start + end) / 2 may arise condition of integer overflow so, be on safer side.

Example:-



(A) Set start, end

(B) Calculate mid = (s+e)/2

start = 0, end = 7

mid = (0+7)/2 = 7/2 = 3

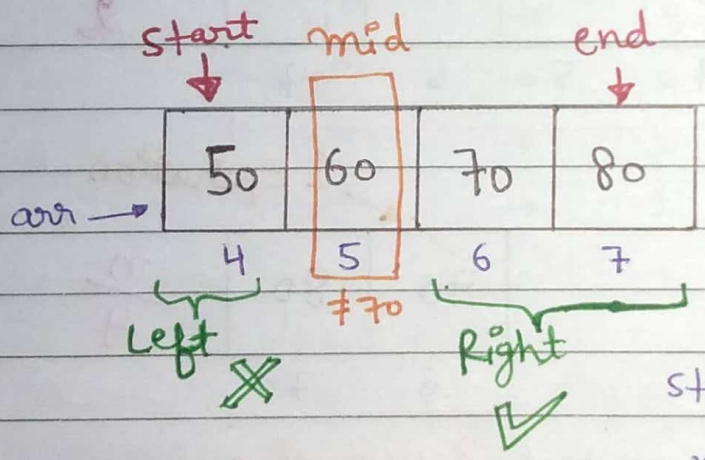
(C) Check the condition

(D) Select one part and neglect the other

40 == 70 → F

70 > 40 → select right side array

↳ Neglect the left side array



start = 4, end = 7

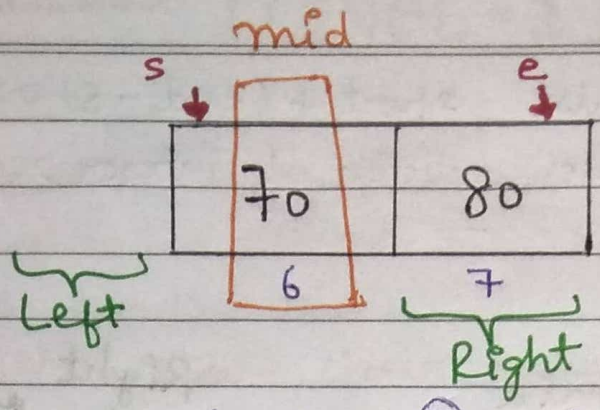
mid = (4+7)/2 = 11/2 = 5

60 == 70 → F

70 > 60 → Select Right

↳ Neglect Left

Date ..../..../.....



(A) Start = 6 , end = 7 (B) mid = $\frac{6+7}{2} = \frac{13}{2} = 6$

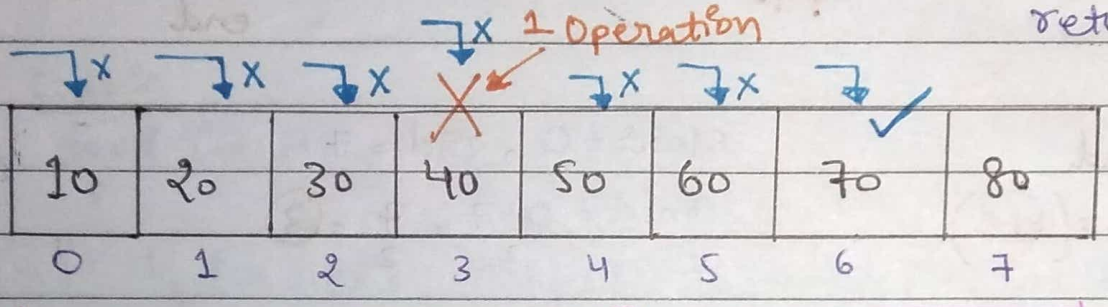
(C) 70 == 70 → Yes → True

↓

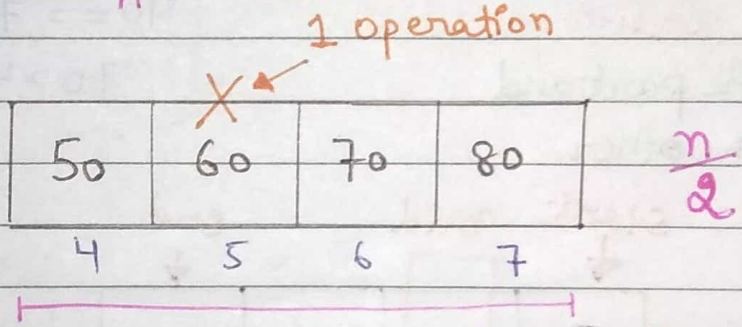
return index

return 6

Working of Binary Search + T.C :-

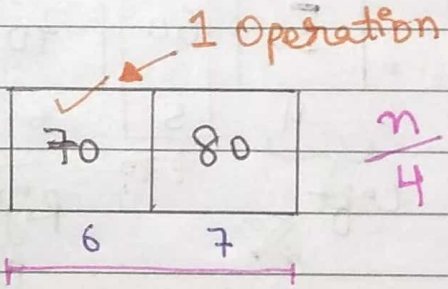


Size = n



★ OBSERVATION :-

With every iteration, array size becomes half in B.S



Binary Search took 3 operations

↳ To tell target found/not found

Linear Search took 7 Search operations, for this specific example.

Date ..../..../.....

Ex:- If we have an array of 10000 elements and we are finding a target which is not present in this array.

If we use Linear Search, in this case it took 10000 comparisons

But Binary Search took very less comparisons
bcz array size becomes half in every iteration.
So,

I st iteration	→ 10000	comparison
II nd	→ 5000	13 comparisons
III rd	→ 2500	
IV th	→ 1250	
5 th	→ 625	
6 th	→ 312	
7 th	→ 156	
8 th	→ 78	
9 th	→ 39	
10 th	→ 19	
11 th	→ 9	
12 th	→ 4	
13 th	→ 2	
14 th	→ 1	

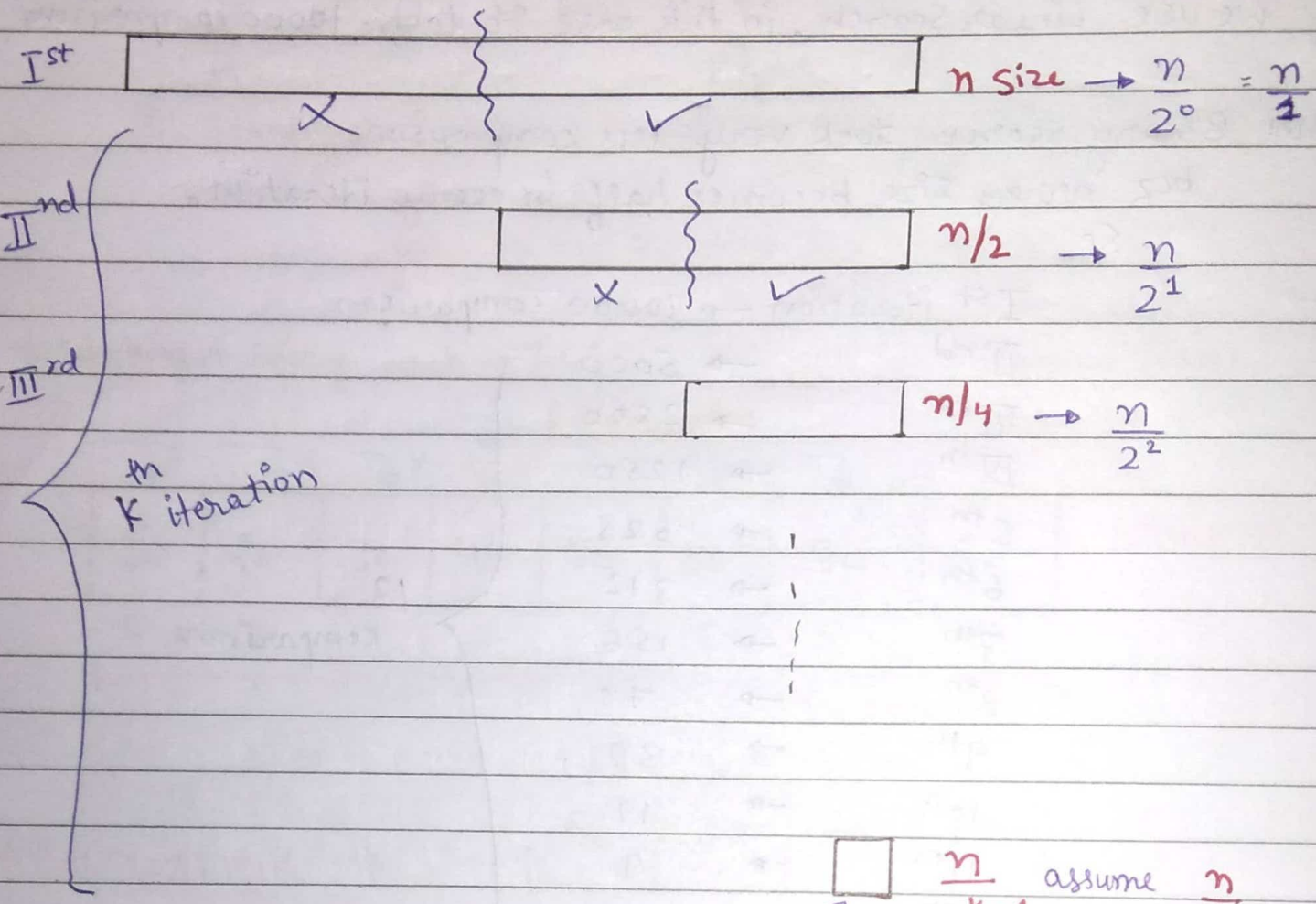
∵ $\log_2(10000)$

Time complexity of Binary Search = $\log_2(n)$

Date ..../..../.....

Time Complexity of Binary Search :-

↳ In every iteration, size becomes half.



∴ We can say that,

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

$$n = 2^k$$

$$\log_2 n = k \quad \text{(Taking log both sizes)}$$

T.C of B.S → $O(\log_2 n)$

$n = 2^k$

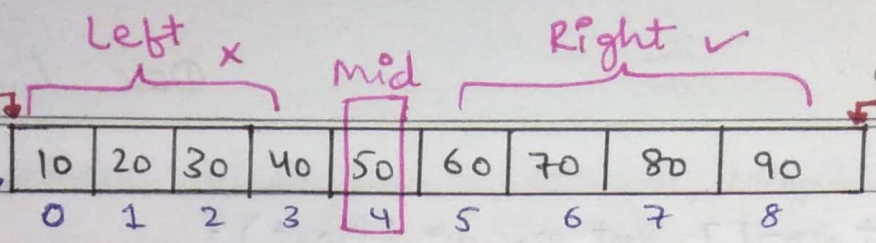
$\log_2 n = \log_2 (2^k)$

$\boxed{\log n = k}$

Spiral

Date ..../..../.....

Ex:- arr

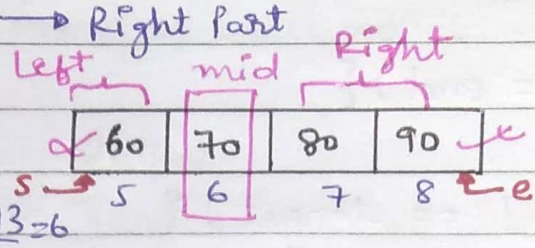


target = 90

$s = 0, e = 8, mid = \frac{0+8}{2} = 4$

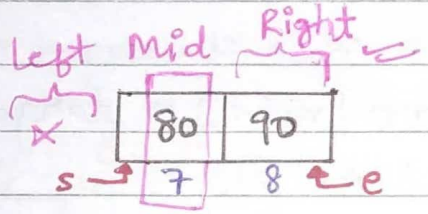
$50 == 90 \rightarrow F$

$90 > 50 \rightarrow$ Right Part



$s = 5, e = 8, mid = \frac{5+8}{2} = \frac{13}{2} = 6$

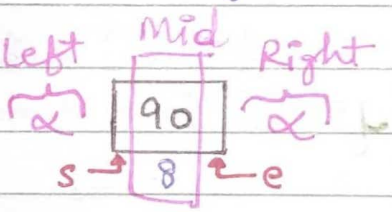
$70 == 90 \rightarrow F \Rightarrow 90 > 70 \rightarrow$ Move towards Right



$s = 7, e = 8, mid = \frac{7+8}{2} = 7$

$80 == 90 \rightarrow F$

$\Rightarrow 90 > 80 \rightarrow$ Move towards Right



$s = 8, e = 8, mid = \frac{8+8}{2} = \frac{16}{2} = 8$

$90 == 90 \rightarrow$ True $\Rightarrow$ Return 8

Rules for applying Binary search:-

① found wala case

```
if (arr[mid] == target)
    return mid;
```

② Not found wale ke case honge

③ ☆ Kabtak apply hoga B.S

while (s <= e)

OR agar (s > e) To bhi Ruk Jao

(i) if (target > arr[mid])

```
s = mid + 1
```

(ii) if (target < arr[mid])

```
e = mid - 1
```

Spiral

Date ..../..../.....

Code:-

```
int binarySearch (int arr[], int n, int target) {
    int start = 0;
    int end = n-1;
    int mid = start + (end - start)/2;

    while (start <= end) {
        // Found
        if (arr[mid] == target) {
            // return index of found element
            return mid;
        }
        else if (arr[mid] < target) {
            // Right me jao
            start = mid + 1;
        }
        else if (arr[mid] > target) {
            // Left me jao
            end = mid - 1;
        }
        // yaha galti nhi Krni hamesha mid update Krna hai
        mid = start + (end - start)/2;
    }
    // Agar yaha tak phoche matlab target nhi mila so
    // return invalid index
    return -1;
}
int main() {
    int arr[] = {10, 20, 30, 40, 50, 60, 70, 80, 90};
    int n = 9;
    int target = 70;
```

Spiral

Date ..../..../.....

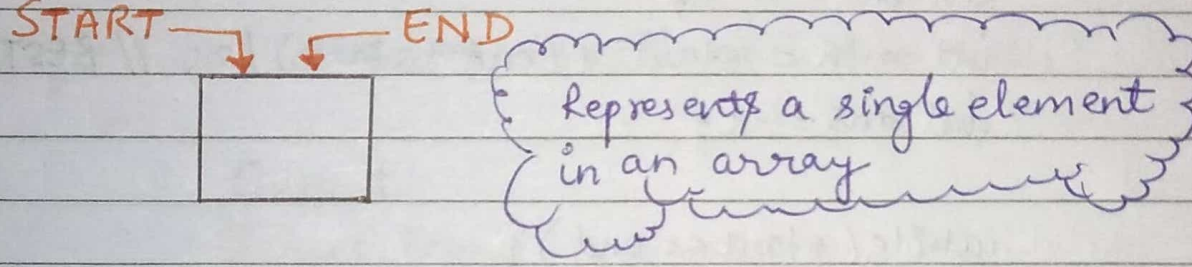
```
int ansIndex = binarySearch (arr, n, target);

if (ansIndex == -1)
        cout << " Target Not found " << endl;
else
        cout << " Target found at index no. : " << ansIndex << endl;
return 0;
}
```

Output

Target found at index no.: 6

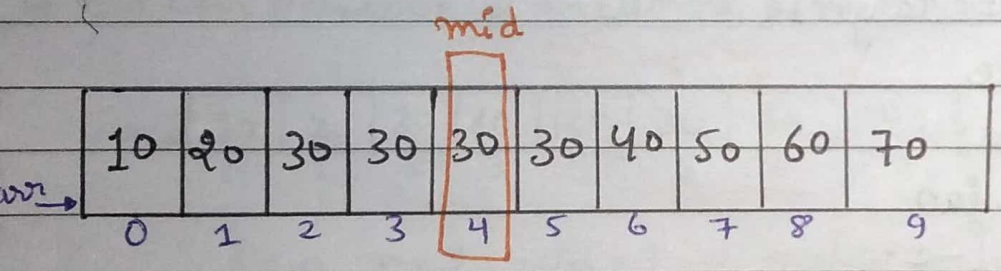
KEEP IN MIND

"START == END" → represents an array whose size is 1.



Q Find first occurence of a number in a sorted array.

Think about B.S if we want to search an item.



target = 30
ans = 2 [Expected Answer]

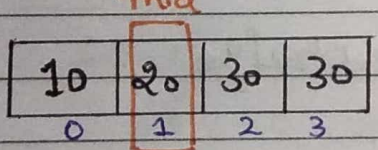
s = 0, e = 9, mid = 4, arr[mid] == target

30 == 30 → True → found

ans = 4

left → e = mid - 1

e = 4 - 1 = 3



s = 0, e = 3, mid = 1

⇒ 20 == 30 → F

30 > 20 → Right → s = mid + 1

s = 1 + 1, s = 2

Rule :-

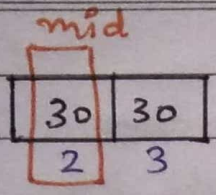
Found kaha

→ ans store then

→ Left me chale jana i.e

end = mid - 1

Date ..../..../.....



s = 2, e = 3, mid = 2

30 == 30 → Found → ans = 2

Left me jao

e = mid - 1

e = 2 - 1

e = 1

s = 2, e = 1

s > e → Rukjao STOP!!

Code:-

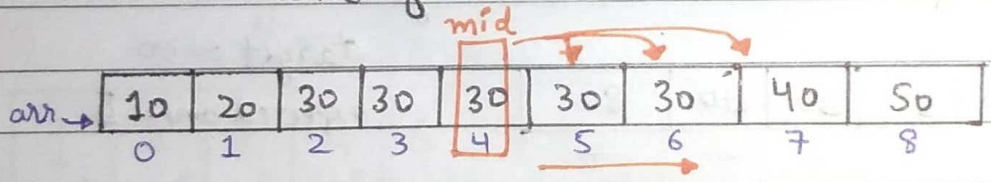
```
int findFirstOccurence(int arr[], int n, int target){
    int start = 0;
    int end = n-1;
    int mid = start + (end - start) / 2; // BESTPRACTICE
    int ans = -1;

    while( start <= end ){
        //found
        if( arr[mid] == target){
            //ans store
            ans = mid;
            // Left me jao
            end = mid - 1;
        }
        else if (arr[mid] > target) {
            // Left me jao bcz 1's occurence chaiye
            end = mid - 1;
        }
        else if (arr[mid] < target) {
            // right me jao
            start = mid + 1;
        }
```

```
    // Galti yaha karte hai hamesha
    mid = start + (end-start)/2;
  }
  return ans;
}
int main(){
  int arr[] = {10, 20, 30, 30, 30, 30, 70, 80, 90};
  int n = 9;
  int target = 30;
  int ansIndex = findFirstOccurence(arr, n, target);
  if(ansIndex == -1)
      cout<<"Target not found"<<endl;
  else
    cout<<"Target found at index no. :"<<ansIndex<<endl;
  return 0;
}
```

Output

Target Found at index no. : 2

Q Find Last occurence of an element in a sorted array.



s = 0, e = 8, mid = 4 => arr[mid] == target

30 == 30 → True → Found → (i) ans store

(ii) Right me jao

↳ s = mid+1

Code:-

```
int findLastOccurence(int arr[], int n, int target){
  int s = 0; int e = n-1; int mid = s+(e-s)/2;
  int ans = -1;
  while(s<=e){
  // Found
  if(arr[mid] == target){
```

```
        // store ans
        ans = mid;
        //right me jao
        s = mid+1;
      }
      else if (arr[mid] < target){
        //right me jao
        s = mid+1;
      }
      else if (arr[mid] > target){
        //Left me jao
        e = mid-1;
      }
      //yaha galtinhi krni yaad rkho update mid
      mid = s + (e-s)/2;
    }
    return ans;
}
```

Q Find total occurence of an element in sorted array.

10	20	30	30	30	30	30	80
0	1	2	3	4	5	6	7

target = 30
Expected ans = 5

First occurence = 2
Last Occurence = 5

Total Occurence = Last Occurence - first Occurence + 1

Code:-

```
int findTotalOccurence (int arr[], int n, int target) {
    int total;
```

Date ..../..../.....

```
int lastOccurence = findLastOccurence(arr, n, target);
int firstOccurence = findFirstOccurence(arr, n, target);
total = lastOccurence - firstOccurence + 1;
return total;
}
int main(){
int arr[] = {10, 20, 30, 30, 30, 30, 30, 40};
int n = 8;
int target = 30;
int total = findTotalOccurence(arr, n, target);
cout << "Total occurence is : " << total << endl;
return 0;
}
```

Output :-
Total Occurence is : 5

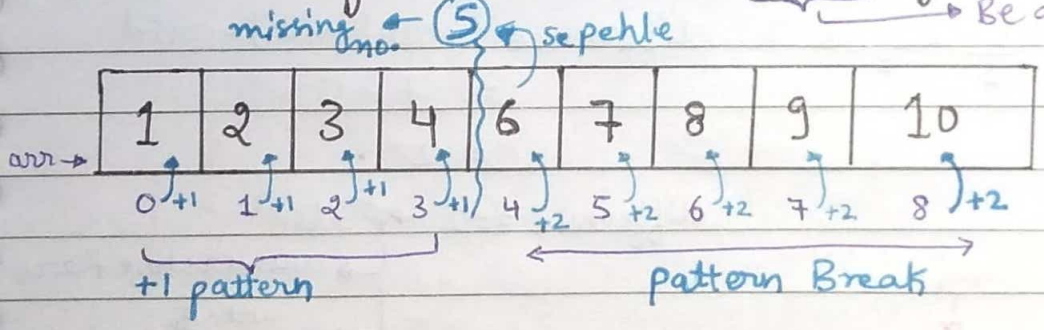
★ Binary Search ke Advance Questions are based upon understanding of firstOccurence and lastOccurence questions. Ki agar FOUND wali condition hai to RIGHT me jao, answer store karo, etc.

Q Find missing element in a sorted array.

→ Be attentive to think about B.S.

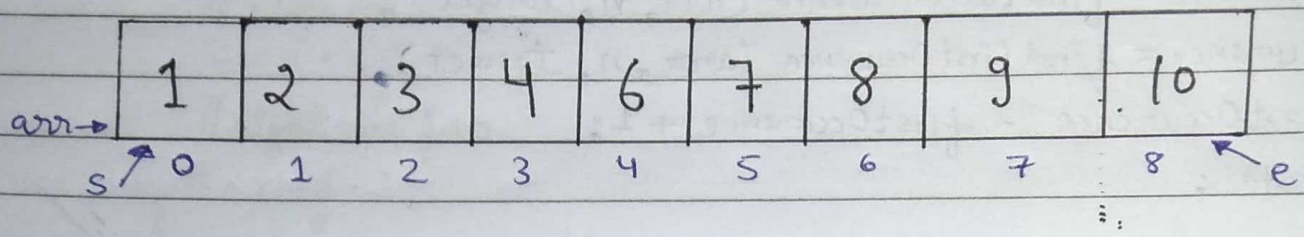
given → 1 to n no.'s

ans = 5 (missing no.)

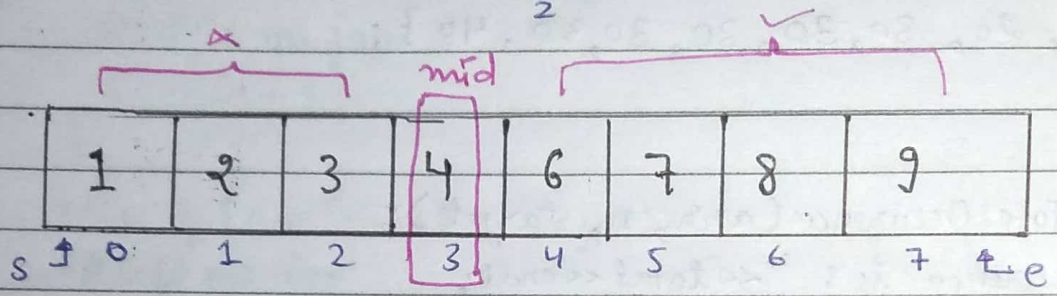


★ Jabtak missing element nhi ayga tabtak +1 pattern chalega, Or jese hi pattern change hoga toh jaha se pattern change hoga uske just pehle wala element hi missing element hoga.

Date ..../..../.....



s=0, e=08, mid = $\frac{0+8}{2}$ =



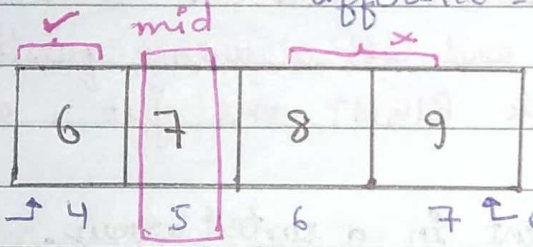
s=0, e=7, mid = $\frac{0+7}{2}$ = 3 difference = arr[mid] - mid

= 4 - 3

= 1

difference == 1 → Right me jao

s = mid + 1



s=4, e=7, mid = $\frac{4+7}{2}$ = $\frac{11}{2}$ = 5 difference = arr[mid] - mid

= 7 - 5

= 2

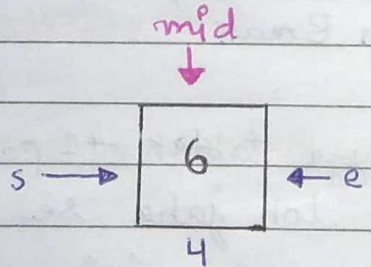
difference == 1 → False

↳ answer store Karo

ans = 5

↳ Left me jao

e = mid - 1



s=4, e=4, mid = $\frac{4+4}{2}$ = 4 difference = arr[mid] - mid

= 6 - 4

Date ..../..../.....

difference = 2

difference == 1 → No → ANS store , ans = 4

↳ LEFT me jao, e = mid-1

s=4, e=3 ⟶ s>e ⟶ Rukjao

Final Ans = ans+1

Code:-

```
int findMissingElement( int arr[] , int n){
 int s=0; int e = n-1; int mid = s+(e-s)/2; int ans=-1;
 while (s<=e){
    int diff = arr[mid] - mid;

    if(diff == 1){
       // right me jao
       s= mid+1;
    }
    else{
       //ans store kro
       ans = mid;
       // Left me jao
       e = mid - 1;
    }
    mid = s+(e-s)/2;
 }
 if(ans +1 == 0){
    return n+1; // for the corner case when missing number is the last no
 }
return ans + 1;
}
```

Date ..../..../.....

Q Peak Index in a Mountain Array. 852 (Leetcode)

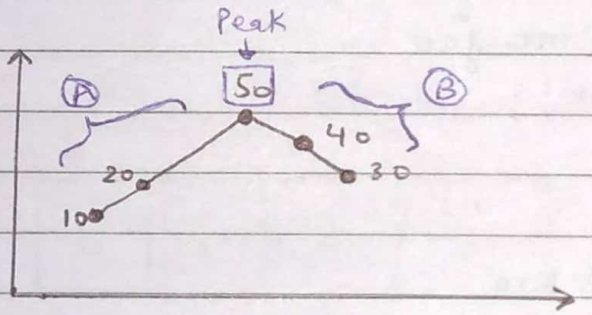
An array 'arr' is a mountain if the following properties hold:

- 'arr.length >= 3'
- There exists some 'i' with '0 < i < arr.length - 1' such that:
  - arr[0] < arr[1] < ... < arr[i-1] < arr[i]
  - arr[i] > arr[i+1] > ... > arr[arr.length+1] > ... > arr[arr.length - 1].

You must solve it in O(log(arr.length)) time complexity.

if array → array[] = {10, 20, 50, 40, 30}

o/p → ans = 50



(A)

arr[i] < arr[i+1]

(B)

arr[i] > arr[i+1]

Peak Point

arr[i-1] < arr[i] > arr[i+1]

∴ If arr[i] < arr[i+1] → Exist in (A)

if Not then it must be Exist in (B) Line or a Peak element

Observation

(A) Line

arr[i] < arr[i+1]

(B) Line

arr[i] > arr[i+1]

Peak

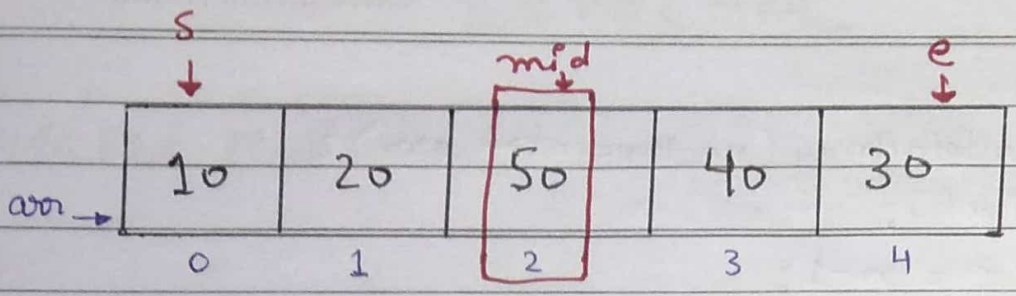
arr[i] > arr[i+1]

arr[i] > arr[i-1]

we can club

Spiral

Date ..../..../.....



s=0, e=4, mid = $\frac{0+4}{2}$ = 2, arr[mid] < arr[mid+1]

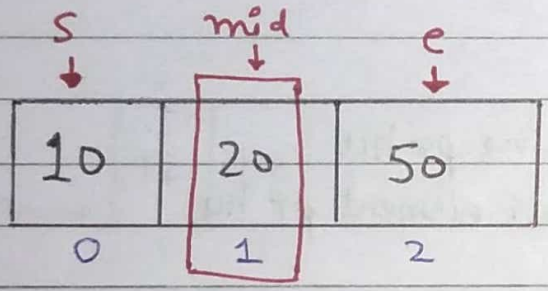
50 < 40 → False

↳ Either (B) or Peak

↳ Left me jao

e = mid - 1 ✗

(If we use this, we lost our peak)



e = mid ✓

(Benefit :- Left bhi chale gaye or peak bhi lost nhi hui)

s=0, e=2, mid = $\frac{0+2}{2}$ = 1, arr[mid] < arr[mid+1]

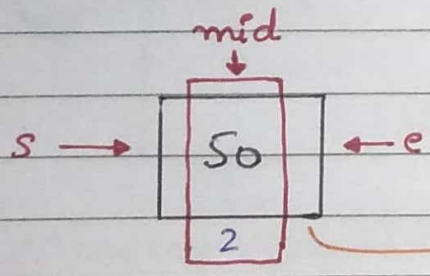
20 < 50 → True → (A) Line Pr hai

↳ Right me jao

s = mid + 1

s = 1 + 1

s = 2



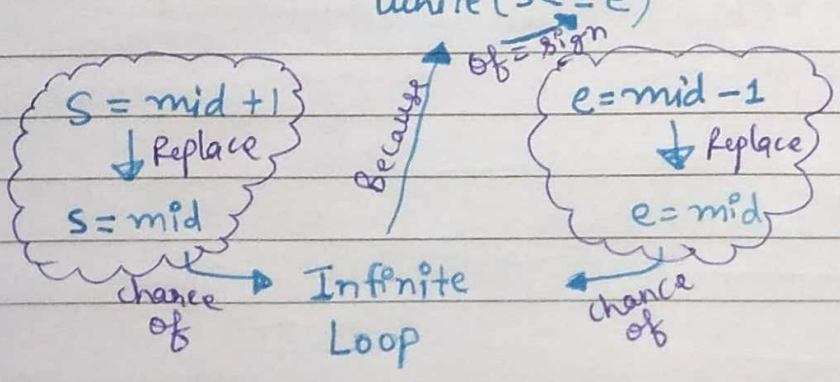
Single Element

↳ its the Peak

return s

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

KEEP IN MIND:-



∴ Remove '=' sign while changing these forward | Backword Conditions

Spiral

Date ..../..../.....

Code :-

```
int peakIndexInMountainArray(vector<int>& arr) {
    int n = arr.size();
    int s = 0; int e = n-1;
    int mid = s+(e-s)/2;
    while(s<e) {  // Infinite Loop se bachne ke liye s<e
        if(arr[mid] < arr[mid+1]) {
            // A wali line me hu
            // Peak right me exist krti hai
            s = mid+1;
        }
        else {
            // Yaa toh main B line pr hu
            // Yaa toh main peak element pr hu
            e = mid;
        }
        // yaha galti nhi krni so update the mid
        mid = s+(e-s)/2;
    }
    return s;
}
```