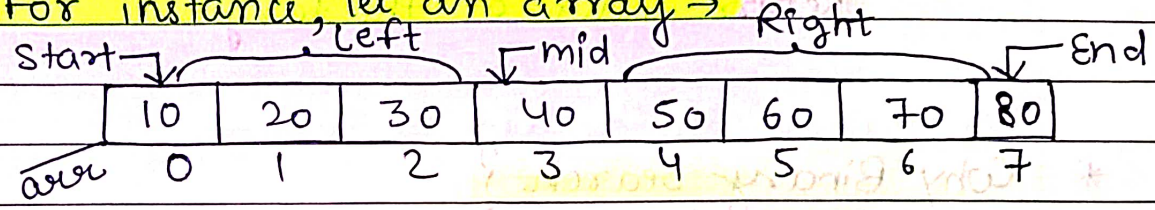
13/09/2023
Wednesday

* BINARY SEARCH →

Binary search can only be applied on sorted array that is it must be monotonically increasing or decreasing.

For instance, let an array →



(S) start = 0 let Target = 70

(e) end = 7

$$mid = s + \frac{(e-s)}{2} = \frac{0+7}{2}$$

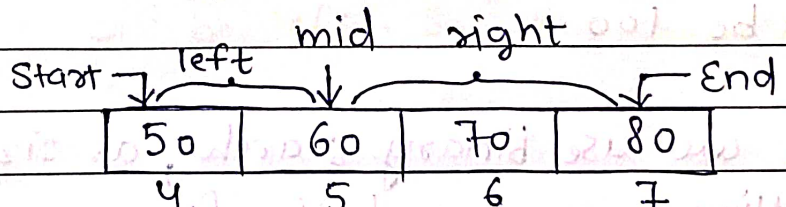
mid = 3 (integral value)

arr[mid] = 40

Target > arr[mid]

70 > 40

AS, it is a sorted array. So, target will must be in the right part.



s = 4 ; e = 7

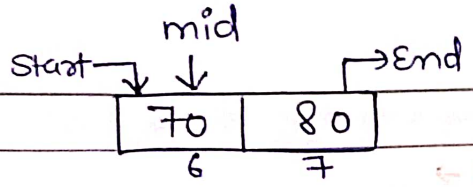
$$m = \frac{7+4}{2} = \frac{11}{2} = 5 \text{ (integral value)}$$

arr[mid] = 60

70 > 60

So, target is in the right part.

__/__/__



s = 6 , e = 7

$m = \frac{6+7}{2} = \frac{13}{2} = 6$ (Integral value)

arr[mid] = 70

Target = arr[mid]

Return index of the target

* Why Binary Search ?

In Binary search, after every iteration, the size of the array gets reduced to the half of before. So, number of operations gets reduced too.
whereas, in linear search, we have to traverse the whole array.

For instance, in an array of size = 100 →

If we use linear search, number of operations would be 100.

And if we use binary search, as size of array is getting reduced to half after every iteration. Number of operation would be 7 only for array of size 100, 50, 25, 12, 6, 3, 1.

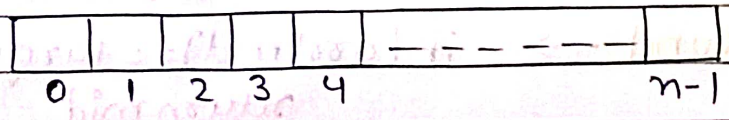
Binary search also reduces the time complexity.

NOTE - Time complexity of Binary Search is →

$$O(k) = O(\log_2 n)$$

* **Time Complexity of Binary Search →**

let an array of size n.



No. of iteration	Size of array
1 →	$\frac{n}{2^0} \rightarrow n$
2 →	$\frac{n}{2^1}$
3 →	$\frac{n}{2^2}$
⋮	⋮
k →	$\frac{n}{2^{k-1}} \simeq \frac{n}{2^k}$

At k^{th} iteration, only single block of the array will be left. So, the size of array will be 1.

Also, the size of array for k^{th} iteration $= \frac{n}{2^k}$

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

$n = 2^k$

$\log_2 n = \log_2 2^k$ (Taking log both sides)

$\log_2 n = k \log_2 2$ $(\because \log_n n = 1)$

$$\boxed{\log_2 n = k}$$

__/__/____

* **Rules for applying Binary Search** → Ascending order

1. The loop will continue until **start <= end.**

2. // found →
```
if (arr[mid] == target)
    return mid
```

3. // not found →

// Shifting to the left of mid →
```
if (target < arr[mid])
    end = mid - 1
```

// Shifting to the right of mid →
```
if (target > arr[mid])
    start = mid + 1
```

4. Remember to **update mid** after every iteration. where mid is $start + \frac{(end - start)}{2}$.

* **Program of Binary Search** →

I/P = 10 20 30 40 50 60 70 80 90
Target = 30

O/P = Target found at index : 2

NOTE→ when start==end, it means size of array gets reduced to 1. Only single element is present there where both start and end are pointing.

__/__/__

Program→

```
#include <bits|stdc++.h>
using namespace std;

int binarySearch(vector<int>&v, int target){
    int s=0;
    int e = v.size()-1;
    int mid= s+(e-s)/2;
    while(s<=e){
//found  if(target==v[mid]){
            return mid;
        }
//not found  else if(target>v[mid]){
            s=mid+1;
        }
        else if(target<v[mid]){
            e=mid-1;
        }
//updating mid  mid = s+(e-s)/2;
    }
    return -1;
}
int main(){
    vector<int>v{10,20,30,40,50,60,70,80,90};
    int target = 30;
    int ansIndex = binarySearch(v, target);
    if(ansIndex==-1){
        cout<<"target not found"<<endl;
    }
    else
    cout<<"target found at index:"<<ansIndex<<endl;
}
```

__/__/____

* Find first occurence →

I/P = 10 20 30 30 30 30 40 50, Target=30

O/P = Target found at index: 2

Logic→ // found → if (target == v[mid]){
ans=mid;
e=mid-1; }

Remaining same as Binary search

Program→
```
#include <bits/stdc++.h>
using namespace std;

int firstOccurence (vector<int>&v, int target) {

    int s=0;
    int e = v.size()-1;
    int mid= s+(e-s)/2;

// ans will store the current value of mid
    int ans = -1;

    while (s<=e){
        if (target==v[mid]) {
            ans = mid;
            e = mid - 1; // for finding first occurence we have to move to the left

    }
```

```
else if (target > v[mid]){
        s= mid+1;
      }
else if (target < v[mid]){
        e = mid-1;
      }
 mid=  s+(e-s)/2;
}
return ans;
}
int main(){
 vector<int>v{10,20,30,30,30,30,40,50};
  int target = 30;
  int ansIndex = firstOccurence(v, target);
  if (ansIndex == -1){
       cout<<" target not found"<<endl;
      }
  else
     cout<<" target found at index: "<< ansIndex<<endl
}
```

* for finding last occurence →

I/P = 10 20 30 30 30 30 40 50

O/P = Target found at index: 5

logic → // found →

```
if (target == v[mid]){
        ans = mid;
        s = mid+1;
      }
```

Remaining same as first occurence.

__/__/____

* Find total occurence →

I/P = 10 20 30 30 30 30 40 50
Target = 30

O/P = total occurence is : 4

Logic→ Find first occurence.
Find last occurence.
Total occurence = last Occurence − first occurence + 1

Program→

```
#include<bits/stdc++.h>
using namespace std;

int firstOccurence (vector<int>&v, int target){
    int s=0; int e=v.size()-1;
    int mid= s+(e-s)/2;
    int ans = -1;
    while (s<=e){
        if (target==v[mid]){
            ans=mid;
            e= mid-1;
        }
        else if (target >v[mid]){
            s= mid+1;
        }
        else if (target <v[mid]){
            e= mid-1;
        }
        mid= s+(e-s)/2;
    }
    return ans;
}
```

```
int lastOccurence (vector<int>&v, int target){
    int s=0; int e= v.size()-1;
    int mid= s+(e-s)/2;
    int ans= -1;
    while (s<=e){
        if (target== v[mid]){
            ans=mid;
            s= mid+1; }
        elseif (target > v[mid]){
            s=mid+1; }
        elseif (target< v[mid]){
            e=mid-1; }
        mid = s+(e-s)/2;
    }
    return ans;
}
int totalOccurence (vector<int>&v, int target){
    int firstOcc= firstOccurence (v, target);
    int lastOcc = lastOccurence (v, target);
    int total = lastOcc- firstOcc +1;
}

int main(){
    vector <int> v{10,20,30,30,30,30,40,50};
    int target= 30;
    int totalOcc= totalOccurence (v, target);

    cout<< "total occurence is: "<< totalOcc<<endl;
```

__/__/__

* Program to find missing number. let the range be 1 → N. Given input elements in array is N-1.

I/P = 1 2 3 4 5 6 7 9

O/P = missing element is : 8

Logic → As array is sorted, we will use binary search.

= If difference → v[mid] - mid == 1
Neglect the left side of mid because the difference between element at that index and the index must always be 1 in the left.
Now, check in the right.
So, s = mid + 1

= Else → difference → v[mid] - mid ≠ 1
Store the mid index in ans variable and shift to the left side of mid index to check whether the above condition i.e. difference → 1 holds true or not in left of mid index as in right, it must be false.
So, ans = mid
e = mid - 1

= Update mid.

= Return ans+1.

= Condition if the last element in range is absent →
As ans is initialized with -1. So, if the last element in range will be absent, the ans will remain -1. So, if (ans + 1 == 0) (∴ ans = -1)
return v.size()+1

Program→

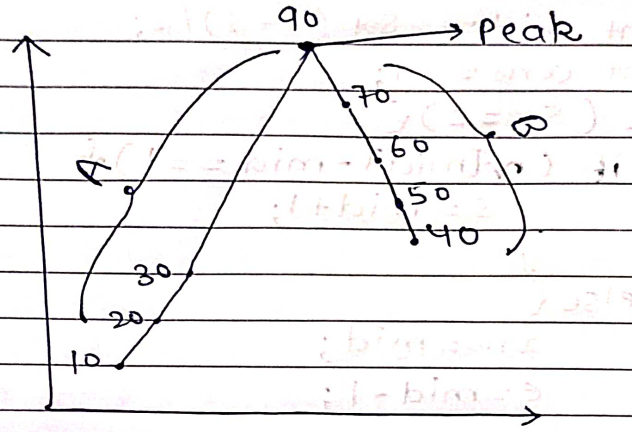
```
#include<bits/stdc++.h>
using namespace std;
int missingElement(vector<int>&v){
    int s=0;
    int e=v.size()-1;
    int mid= s+(e-s)/2;
    int ans=-1;
while(s<=e){
    if(v[mid]-mid==1){
        s=mid+1;
    }
    else{
        ans=mid;
        e=mid-1;
    }
    mid= s+(e-s)/2;
}
if(ans+1==0){
    return v.size()+1;
}
return ans+1;
}
int main(){
    vector<int>v{1,2,3,4,5,6,7,9};
    int missing= missingElement(v);
    cout<<"missing element is:"<<missing<<endl;
}
```

__/__/____

* Peak Element in mountain Array →

I/P = 10 20 30 90 70 60 50 40

O/P = Peak element is at index : 3



= Observations →

In PART A, arr[i] < arr[i+1]

In PART B, arr[i] > arr[i+1]

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

= When we are at any element in part A, move to the right to approach peak element.

= When we are at any element in part B, move to the left to approach peak element.

= When at peak, stay there.

NOTE - In while loop here, the condition is (s<e), if we take it s<=e, it will stuck into infinite loop.

__/__/__

logic →

```
if (arr[mid] < arr[mid+1])  ⟶ (A)
    {
      // right
       s = mid+1;
    }
else {  ⟶ (B) or (peak)
      // left
       e = mid;
    }
```

Program →

```
#include<bits/stdc++.h>
using namespace std;

int peakInMountainArray (vector<int>&v) {
    int n= v.size()-1;
    int s = 0 ; int e= n;
    int mid = s+(e-s)/2;
    while (s<e){
        if (v[mid]<v[mid+1]){
            s=mid+1;
        }
        else{
            e=mid;
        }
        mid= s+(e-s)/2;
    }
return s;
}
int main(){
    vector<int>v{10,20,30,90,70,60,50,40};
    int peak = peakInMountainArray(v);
    cout<<"peak element is: "<<peak<<endl;
}
```