17/feb/23

Lec-10 Searching & sorting - I

Linear Search

arr, key = 7

0	1	2	3	4	5	6
1	22	13	14	50	67	7

n = 7

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

$T.C = O(n)$ ← linear Time complexity

In worst case, this algo takes n comparison to find elements.

In Best Case O(1)

let n = 100000
↳ In worst case we have to do 100000 comparison.

⊙ Binary Search - A type of Searching algo. (optimized).

condition - Elements should be in monotonic order.
(either increasing or decreasing)

target = 15

0	1	2	3	4	5	6	7
1	3	7	9	11	13	15	19
start							end

Binary Search Steps-

① Initialize two pointer start and end.

② Start with 0 and end with the last index (size-1).

② Find mid ~~element~~ index.

$\frac{start + end}{2}$

③ Compare target with mid element

```
if (arr[mid] == target)  ← Element found
if (arr[mid] < target)   ← search in right part
if (arr[mid] > target)   ← search in left part.
```

↳ start = 0, end = 7   mid = $\frac{0+7}{2}$ = 3

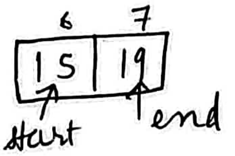
arr[3] = 9 , 9 < 15
arr[mid] < target → right part

4	5	6	7
11	13	15	19
start			end

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

arr[5] = 13

=> 13 < 15 ← right part



start = 6 , end = 7 , mid = $\frac{6+7}{2}$ = 6

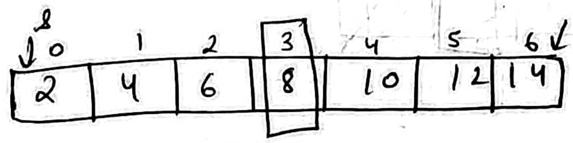
arr[6] = 15

15 == 15 element found at 6 index.

```
int binarySearch(int arr[], int size, int target){
    int start = 0;
    int end = size - 1;
    int mid = (start + end)/2;
    while (start <= end){
        int element = arr[mid];
        if (element == target)
            ~~end = mid + 1~~; return mid;
        else if (target < element)
            end = mid - 1;
        else
            start = mid + 1;

        mid = (start + end)/2;
    }
    return -1;
}
```

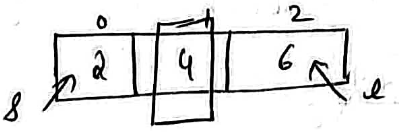
dry run



, target = 2

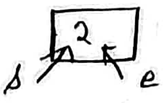
mid = $\frac{0+6}{2}$ = 3 2 == 8 → False

2 < 8 → True (search in left)



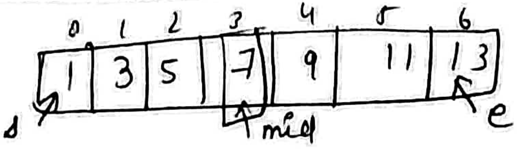
mid = $\frac{0+2}{2}$ = 1 4 == 2 → F

2 < 4 → T (search in left)



mid = $\frac{0+0}{2}$ = 0 2 == 2 → T

(found in 0 index)



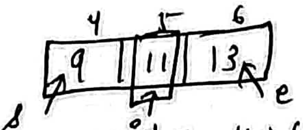
, target = 13

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

7 = = 13 → F

~~9~~13 < 7 → F

↳ search in right part

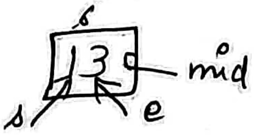


mid = $\frac{4+6}{2}$ = 5

13 = = 11 → F

13 < 11 → F

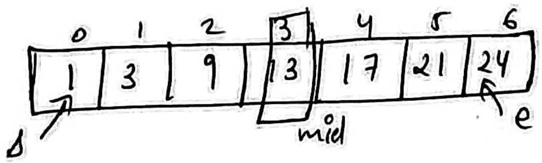
↳ search in right part



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

13 = = 13 → T

Element found in 6 index.



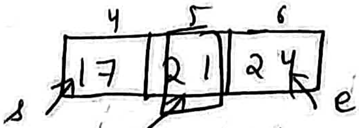
, target = 22

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

22 = = 13 → f

22 < 13 → F

↳ searching in right



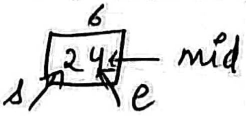
mid = $\frac{4+6}{2}$ = 5

~~↳ see~~

22 = = 21 → f

22 < 21 → F

↳ search in right



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

22 = = 24 → F

22 < 24 → T (search in left)

s = 6 ✓ , e = 6 − 1 = 5 ✓ s > e → stop the loop element not found return -1.

⊙ Issue in mid = (s+e)/2;

let say → $s = 2^{31} - 1$
$e = 2^{31} - 1$

(s+e) ← This will be out of range of integer.

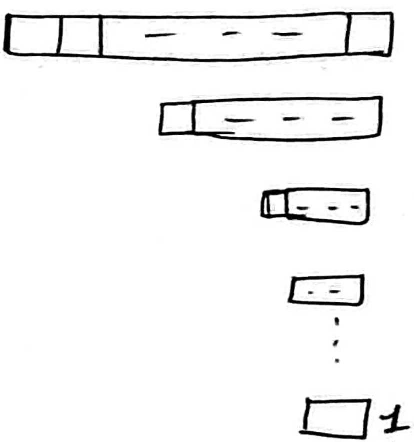
So the condition of integer overflow will occur.

So rather than use this formula we will use this →

mid = s+ (e-s)/2;

⊙ Find out issue in $\frac{s}{2} + \frac{e}{2}$.

T.C →



n size $= \frac{n}{2^0}$

n/2 size $= \frac{n}{2^1}$

n/4 size $= \frac{n}{2^2}$

n/8 size $= \frac{n}{2^3}$

1 = $n/2^k$ size

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

$$n = 2^k$$

Taking log

$$\log n = \log_2 2^k$$

$$\log n = k \log_2 2 \qquad (\log_2 2 = 1)$$

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

$$\Rightarrow \boxed{T.C = O(\log n)}$$

⊙ Binary search in STL →

Must include algorithm library.

```
# include <algorithm >
# include < iostream >
using namespace std;
int  main (){

        vector <int> arr { 2 ,4, 8 , 10, 12 };      // In vector
        int target = 8 ;
        if ( binary_search (arr.begin() , arr. end() , target ))
                cout << "found " ;
        else
                cout << " Not found" ;

        // In Array
        int arr[ ] = { 1 ,2, 3 , 8 , 10 } ;
        int target = 10;
        int n = 6 ;
        if ( binary_search (arr , arr+n , target))
                cout << "Found " ;
        else
                cout << "Not Found";

}
```

<u>Questions</u> -

<u>Ques 1</u> find the first occurence of an element.

i/p →

0	1	2	3	4	5	6	7	8	9
1	3	4	4	4	4	4	6	7	9

, target = 4

o/p → 2

The array is monotonic , so we can apply binary search.

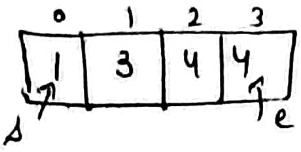
0	1	2	3	4	5	6	7	8	9
1	3	4	4	4	4	4	6	7	9
s									e

$mid = \frac{0+9}{2} = 4$

arr [mid] = = target → True.

4 = = 4

Two Steps
- → store the index as ans.
- → search in left (There is a possibility that the element can exist before mid index too.)



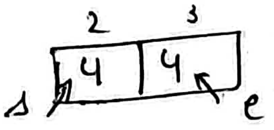
Bruteforce sol^n for this → Linear search from 0th index if find the target, return that index.

T.C = O(n)

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

3 == 4 → F

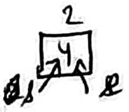
3 < 4 → T
↳ search in right



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

$\boxed{\cancel{4}\ 2}$ ans ✓ ~~ans~~ return ans

4 == 4 → T
↳ update ans.
↳ search in left.



$mid = \frac{2+2}{2} = 2$ ⟶ 4 == 4 → T
↳ update ans
↳ search in left

s = 2 , e = -1
s > e
↳ stop

```
main(){
    vector <int> v{ 1, 3, 4, 4, 4, 4, 6, 7 };
    int target = 4;
    int indexOfFirstOccurenee = firstOccurence (v, target);
    cout << IndexOfFirstOccurence ;
}
int FirstOccurence( vector<int> & arr, int target){
        int s = 0;
        int e = v.size() -1;
        int mid = s + (e-s)/2;
        int ans = -1;
        while (s <= e){
            if (arr [mid] == target){
                    ans = mid;
                    e = mid -1;
            }
            else if (target < arr [mid]
                    e = mid - 1;
            else if ( target > arr [mid]
                    s = mid + 1;
            mid = s + (e-s)/2;
        }
        return ans;
}
```

Last Occurence →

0	1	2	3	4	5	6	7
2	5	7	7	7	7	9	20

s e

target = 7

ans ~~7~~ 5

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

7 = = 7 ⟶ T
↳ ans store
↳ search in right.

4	5	6	7
4	7	9	20

s e

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

7 = = 7 → T
↳ ans store
↳ search in right.

6	7
9	20

s e

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

9 = = 7 → No

7 < 9 → T
↳ search in left.

6
9

s e

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

9 = = 7 → F

7 < 9 → T
↳ search in left.

s = 6 , e = 5 stop and return ans.

Brute force → Linear Search.
Traverse the array from last (size - 1) index. if found target return that index.
TC → O(n).

code -

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

We have upperbound and lowerbound functions in stl to find find last occ. and first occurences.

```
auto ~~last~~ firstOcc = lower_bound (v. begin (), v.end (), target);
auto   lastOcc = upper_bound ( v.begin (), v. end (), target);
cout << firstOcc <<  "  "  << lastOcc << endl;
```

Total Occurence →

i/p →

0	1	2	3	4	5	6	7	8	9
2	4	4	4	4	4	4	6	8	10

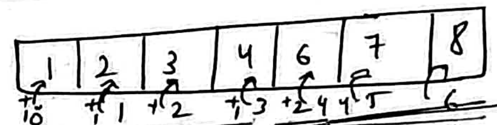
o/p = 6

firstOcc = 1
lastOcc = 6

totalOcc = lastOcc − firstOcc + 1
= 6 − 1 + 1 = 6

⊙ Smallest Missing Number →

⊙ Find Missing element →   1 to n



pattern index+1 = element

pattern break~~ed~~.

H.W → How can we do this ~~u~~using binary search.

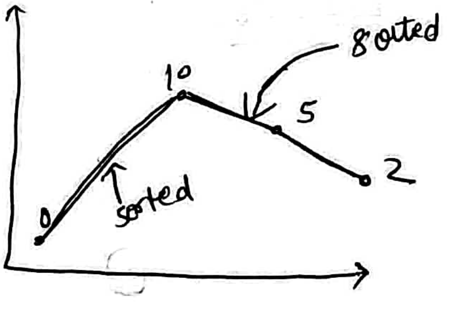
One Approach is → Subtract sum of all the elements of the array from sum of natural no. from 1 to n. Result will be that missing element.

⊙ Peak Element in a Mountain Array →

i/p →

0	1	2	3
0	10	5	2

o/p = 10



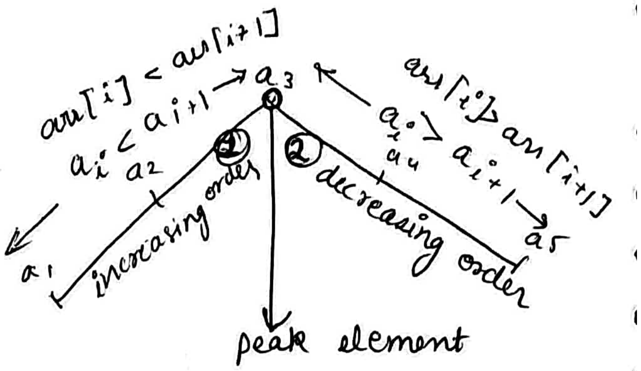
Brute force - linear search → find the maximum element.
O(n) T.C

By Binary Search- Here we have an monotonic array.

0 to 10 → increasing order
10 → 2 → decreasing order.

peak element → A element which is greatest than all the elements.

a_1	a_2	a_3	a_4	a_5

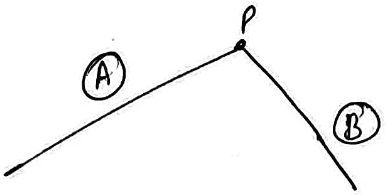


arr[i-1] < arr[i] > arr[i+1]

~~if (arr [mid] > arr[mid+1]~~

↳ line ②

~~if (arr [mid] > arr[mid+1])~~



Line Ⓐ arr[i] < arr[i+1]

P → peak el. → arr[i-1] < arr[i] > arr[i+1]

Line Ⓑ arr[i] > arr[i+1]

if (arr[mid] > arr [mid +1])

→ Two conditions

① ↳ Mid element may be peak el.

② ↳ mid el is in line B.

if (arr [mid] < arr [mid - 1])

↳ arr [mid] can't be a peak el because it is less than an el.

Search in right.

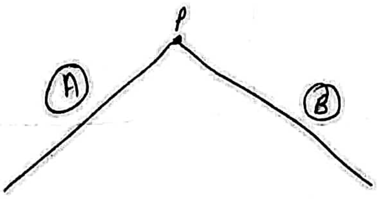
s = mid + 1

else

e = mid;

leetcode ~~LC2~~ 852-

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



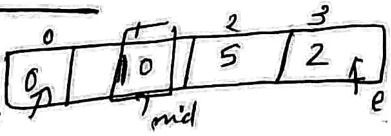
```
if(arr[mid] < arr[mid+1]){   we are at line A. This el. can't be peak.
    s=mid+1;   ↳ search in right
}
else{ ← mid या तो Peak el. पर है या ~~pe~~ B line पर,
    e= mid;
}
```

Now suppose we (mid) are at peak element and we apply e= mid-1; → In this case we will loose the peak el.

So that's why we applied e= mid.

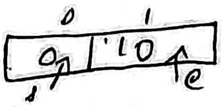
So this e = mid condition can lead us to infinite loop. That's why the while loop's condition is while(s<e).

dry run -



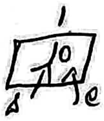
$mid = \frac{0+3}{2} = 1$

s<e
0<3
↳ 10 < 5 → F
e = mid



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

0 < 10 → F
~~s~~= mid +1



s = 1
e = 1
s = = e
stop.
return s (or we can return e too).