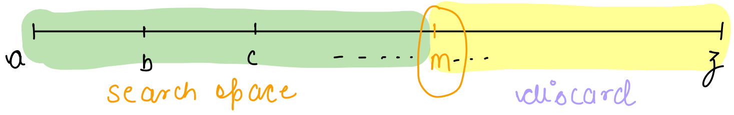
Lecture: Binary search 1

Search story

bro| sis → Police
- → target = bro| sis
- → search space = where to search

Word : [dict, book, newspaper]



Search for bowl.

If search space is sorted| ordered. searching can become easy.

Binary search

Efficient way of searching any el from search space

Neglect one half of search space using any condition

- → Target
- → search space
- → condition

Imp

Q Given arr[n] — sorted and has distinct elements.

Search an el K and return its idx. If el is not found, return -1.

Brute force: Linear search : O(n)

└ Traverse the array

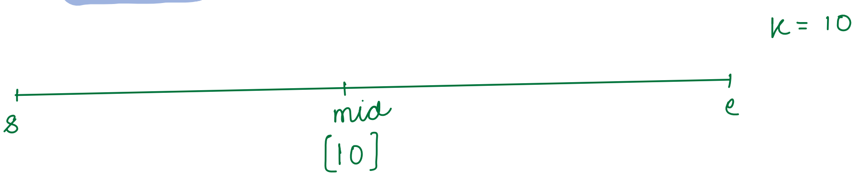
Approach2

target = K

search space = array

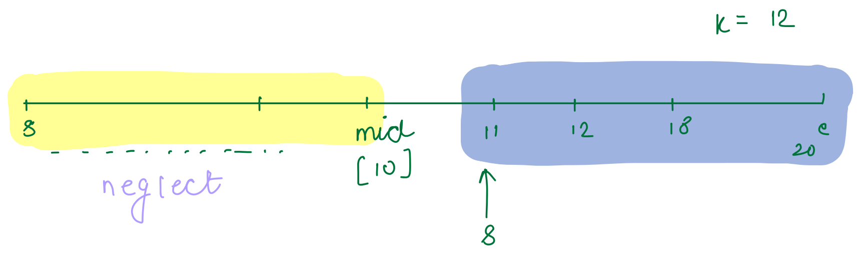
condition

1.



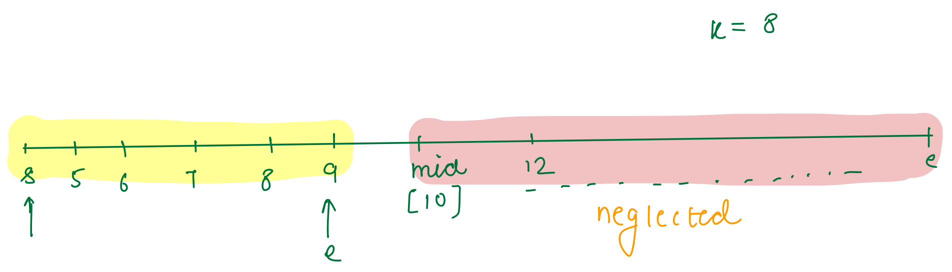
if arr[mid] == K , return mid

2.



s = mid + 1.

3.



e = mid - 1

Dry run:

arr[] = [

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	25	27

]

k = 12

s	e	mid		
0	9	4	arr[4] > k	; e = mid-1
0	3	1	arr[1] < k	; s = mid+1
2	3	2	arr[2] < k	; s = mid+1
3	3	3	arr[3] == k	return 3

```
int bsearch(int[] arr, int k) {
    int s = 0;  e = arr.length-1;
    while( s<=e) {
        mid = (s+e / 2);

        if(arr[mid] < k)
            s = mid+1;

        } else if (arr[mid] >k) {
            e=mid-1;

        } else {
            return mid;

        }
    }
    return -1;
}
```

h|w

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

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

$= \frac{e}{2} + \frac{s}{2} = \frac{1}{2}(s+e)$

TC: n ⟶ n|2 ⟶ n|4 ⟶ n|8 - - - - 1 $O(\log_2 n)$

SC: O(1)

Qu2 Given arr[n] – sorted
find first occ of any el k.
If el is not found, return -1.

Sol: $$\text{arr}[\,] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 3 & 3 & 3 & 4 & 5 & 6 & 6 \end{array}\right]$$

k = 3 → fir occ of 3 = 2.

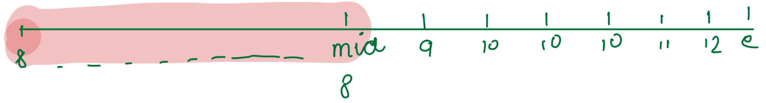
Brute force: Linear search: O(n)

Approach 2: target = fir occ of k.

search space = array

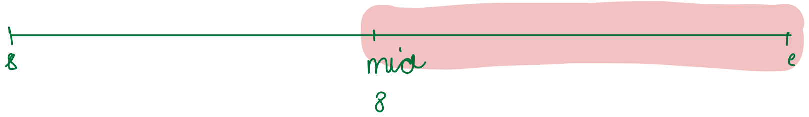
Condition

1. k = 10



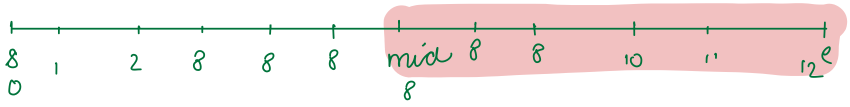
s = mid+1.

2. k = 5



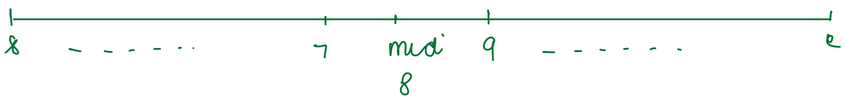
e = mid-1.

3. k = 8



ans = mid

e = mid-1



Dry run:

arr[] = [

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

]

k = 5

s	e	mid	

0	18	9	arr[9] == k ;
			ans = 9
			e = mid − 1

0	8	4	arr[4] < k s = mid + 1
5	8	6	arr[6] < k s = mid + 1

7	8	7	arr[7] == k
			ans = 7
			e = mid − 1

7	6	⟶	break

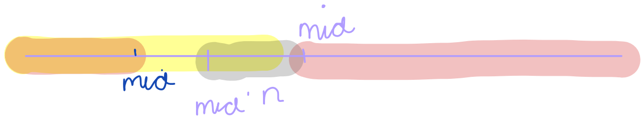
```
int   findfirstoccurence(int[] arr, int k)
      int s = 0;   e = arr.length-1;   ans = -1
      while( s<=e) {
           mid = (s+e / 2);
           if( arr[mid] < k)
                s = mid+1;
           } else if (arr[mid] >k) {
                e=mid-1;
           } else {
                ans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

h|w

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

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

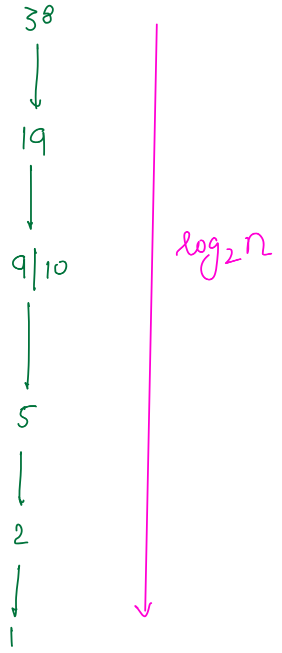
$= \frac{e}{2} + \frac{s}{2} = \frac{1}{2}(s+e)$



TC: $O(\log_2 n)$

SC: $O(1)$

n = 38.



Qu3: Given arr[n] — unsorted

Every el appears twice but for one element.

Note: Duplicate el will be adjacent to each other

find unique el.

Sol: arr: [3 3 1 1 2 5 5 7 7]

Approach1: xor of whole array TC: O(n)

Approach2:

arr[] = [

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

]

1st occ of any number is at even idx

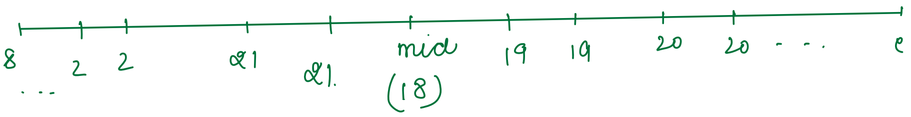
1st occ of any el is at odd idx

Target = unique element

Search space = whole array

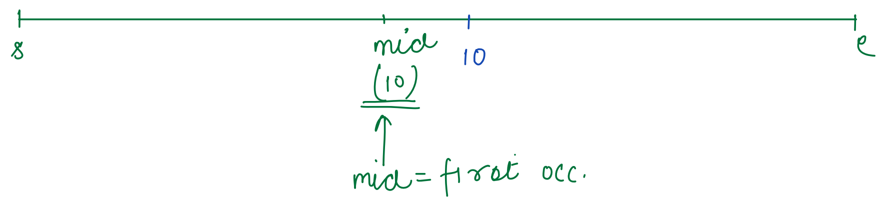
Condition:

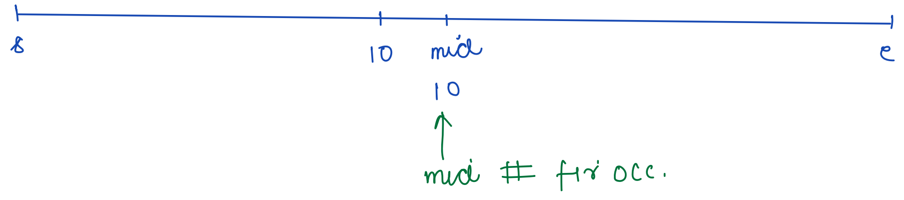
1.



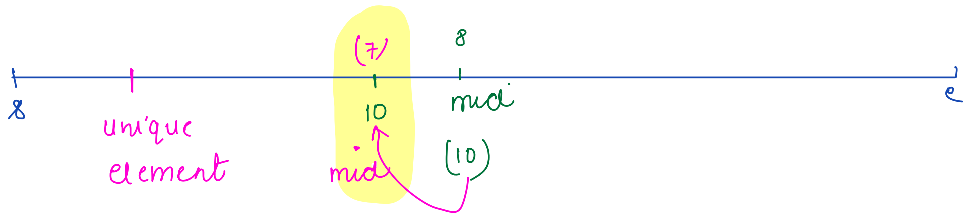
if(arr[mid] != arr[mid-1] &&
arr[mid] != arr[mid+1]) → return mid;

2.





```
if(arr[mid] == arr[mid-1]){   ≠ first occ^n
    mid = mid-1;   // always denote first occ
}
```



```
if (mid % 2 != 0){
    // go to left
        e = mid-1;
} else{
    s = mid+~~1~~; 2
}
```

Dry run:

arr :-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

s	e	mid	
0	14	7	$arr[\overset{mid}{7}] == arr[mid-1] \Rightarrow mid = mid-1.$ $mid = 6$ $mid \% 2 == 0$; $s = mid + 2$
8	14	11	not unique $arr[mid] \;!= arr[mid-1]$ $mid \% 2 \;!= 0$ $e = mid-1$
8	10	9	not unique $arr[mid] \;!= arr[mid-1]$ $mid \% 2 \;!= 0$ $e = mid-1$
8	8	8	unique el. return 8.

```
int    findUnique( int [] arr) {
↑
idx       int s = 0 ;
          int  e =  arr.length -1;
          if ( n==1)    { return  0;}
          if( arr[0] != arr[1]) { return  0;}
          if( arr[n-1] != arr[n-2]) { return  n-1;}

          while( s<=e ){
               mid =  s + ((e-s)/2);

               if ( arr[mid] != arr[mid-1]         &&
                      arr[mid] != arr[mid+1]) {
                         return  mid;
               }
               if ( arr[mid]  ==  arr[mid-1]){
                      mid = mid-1;
               }
               if (mid % 2  != 0){
                      // go to left
                         e = mid-1;
               } else{
                      s = mid +2;
                 }
          }
     }
   return -1;
}
```

TC: $O(\log_2 n)$
SC: $O(1)$

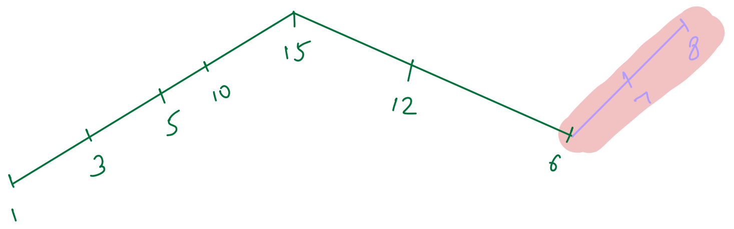
Break: 8:47 AM.

Qu Peak element:

Given inc dec array with distinct el.

Inc dec array: [1 3 5 10 15 12 6]

find max el in array.



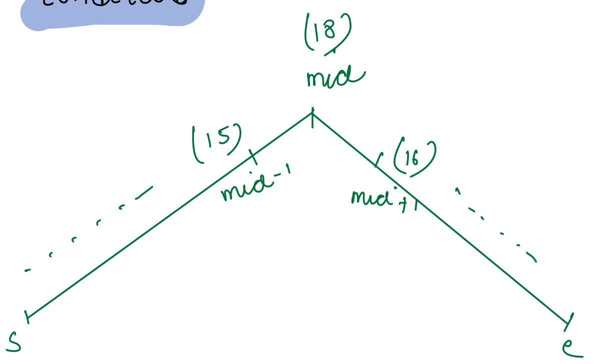
Brute force: Linear traversal O(n)

Approach2: Target = max el of array

search space = arr.

conditions

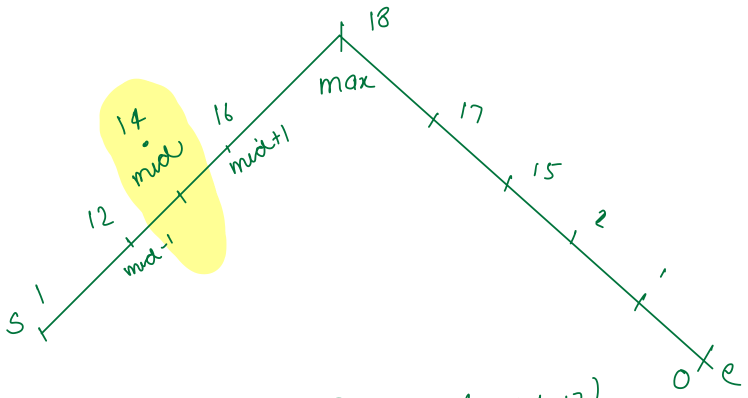
1.



if (arr[mid] > arr[mid-1] &&

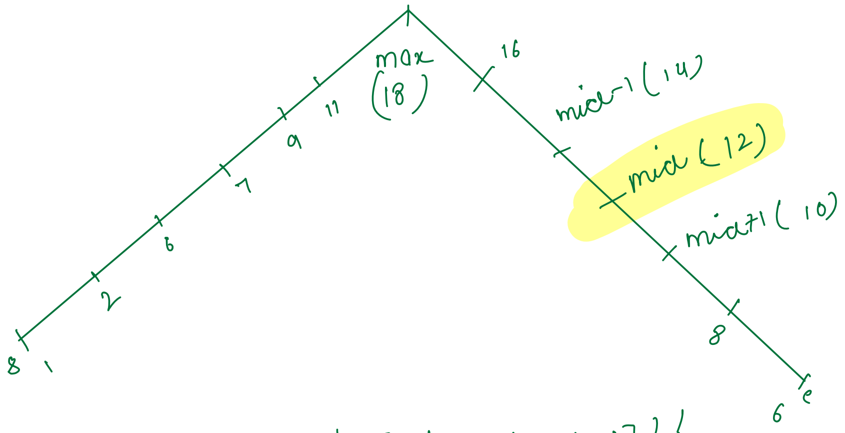
arr[mid] > arr[mid+1]) return mid;

2.



```
if ( arr[mid] > arr[mid-1])
        s = mid +1;
```

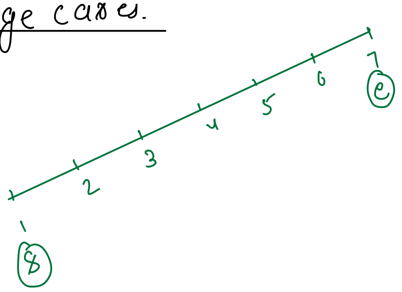
3.



```
if (arr[mid] < arr[mid-1]) {
        e = mid-1;
}
```

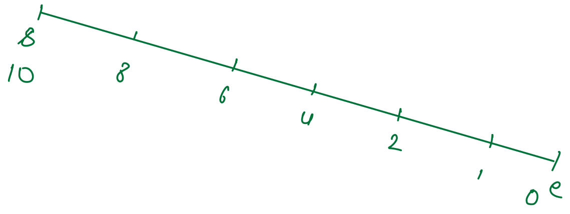
Edge cases:

1.



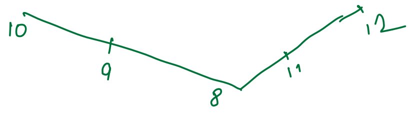
→ return n-1.

2.



→ return 0;

3. 10 9 8 11 12



Invalid

Dec , inc r.

4. arr: [10] return 0th idx.

TC: $O(\log_2 n)$

SC: $O(1)$

Assignments : code it out.

Qu. Local minima

Given arr[n] distinct elements, find any local minima in the arrays.

Local minima:- any no that is smaller than its adjacent neighbours.

arr: [3 6 1 9 15 8] → ans = 1

arr: [21 20 19 17 15 9 7] → ans = 7

arr: [5 8 12 3] → ans = 5

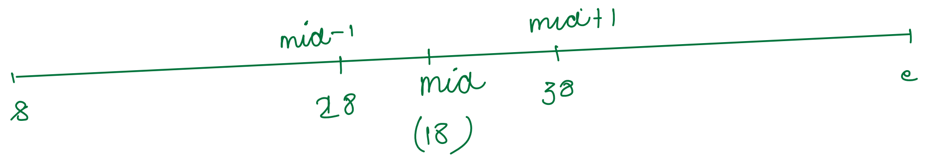
Brute force: Linear traversal. O(n)

Approach2: target = any local minima

search space = whole array

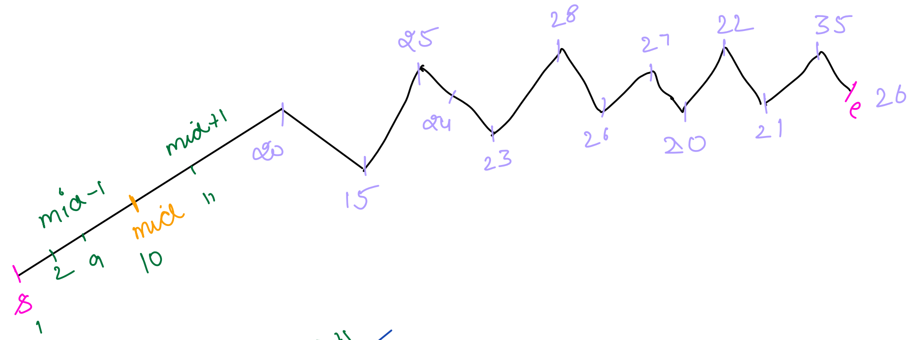
conditions

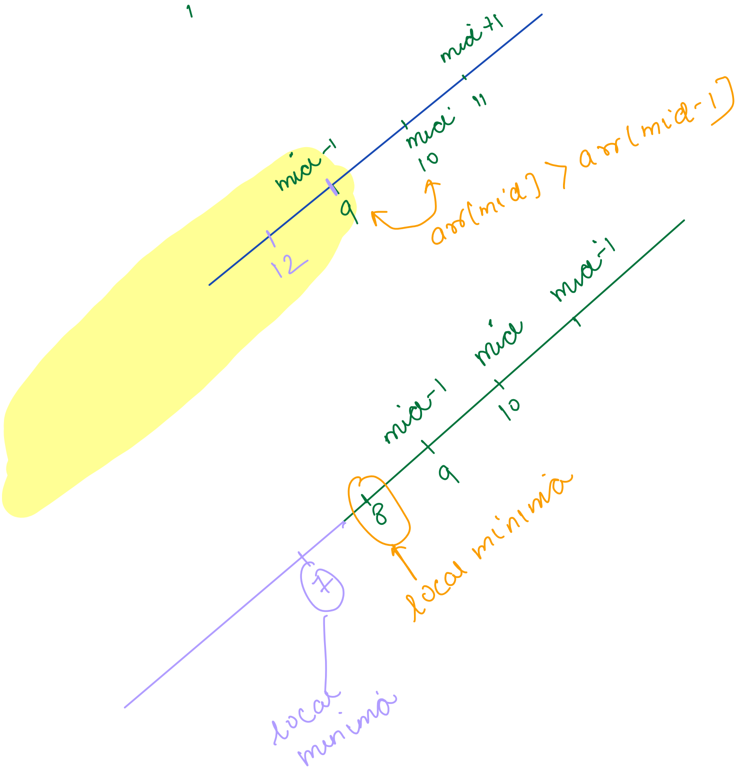
1.



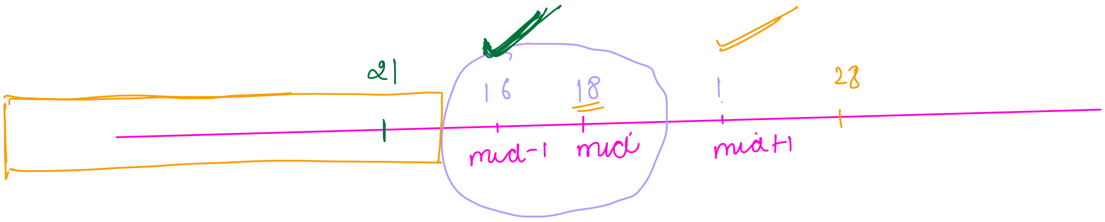
```
if (arr[mid] < arr[mid-1] &&
    arr[mid] < arr[mid+1]) {
        return mid;
}
```

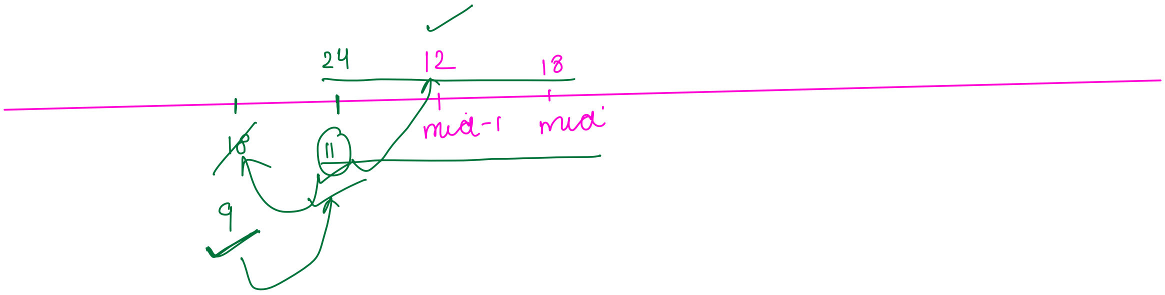
2.



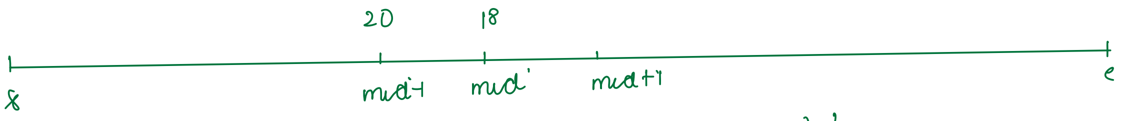


```
if ( arr[mid] > arr[mid-1]) {
    // left
    e = mid-1;
}
```





3.



```
if ( arr[mid] < arr[mid-1] ) {
        // right
        s = mid+1;
}
```

idx

```
int findLocalMinima(int[] arr) {
        int n = arr.length;
        if (n==1) { return 0; }
        if (arr[0] < arr[1]) { return 0;}
        if (arr[n-1] < arr[n-2]) { return n-1;}

        int s = 1;
        int e = n-2;
        while( s<=e) {
              mid = s + ((e-s)/2);
              if ( arr[mid] < arr[mid-1]  &&
                   arr[mid] < arr[mid+1]) {
                   return mid;
              } else if ( arr[mid] > arr[mid-1]) {
                   e = mid-1;
              } else {
                   s = mid+1;
              }
        }
    return -1;
}
```

TC: $O(\log_2 n)$
SC: $O(1)$

Open-ended question

$mid = s + \left(\frac{e-s}{2}\right)$

int s, int e

int $mid = \left[\frac{s+e}{2}\right]$ —— chances of overflow

s = Integer.Max-value

e = Integer.max_value

$(s + e) \longrightarrow$ overflow int

$mid = s + \left(\frac{\boxed{e-s}}{2}\right)$ —never be overflo

$s = \infty$ Int.max

$e = \infty$ Int.max

$s \quad \infty + \left(\frac{(\infty - \infty)}{2}\right)$

$\boxed{\infty + 0}$ = never overflow

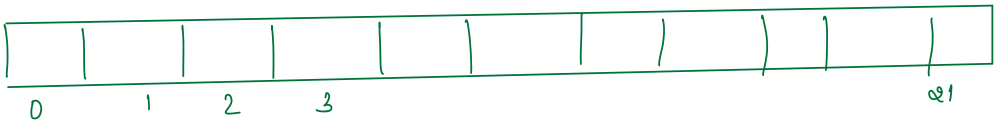
Thankyou ☺

Doubt

arr[n]

$-10 <= arr[i] <= 10$ [21]

freq :-



0 → -10

1 → -9

2 → -8

3 → -7

⋮

21 → 10

logic [Additional work of mapping -ve no to +ve indices]