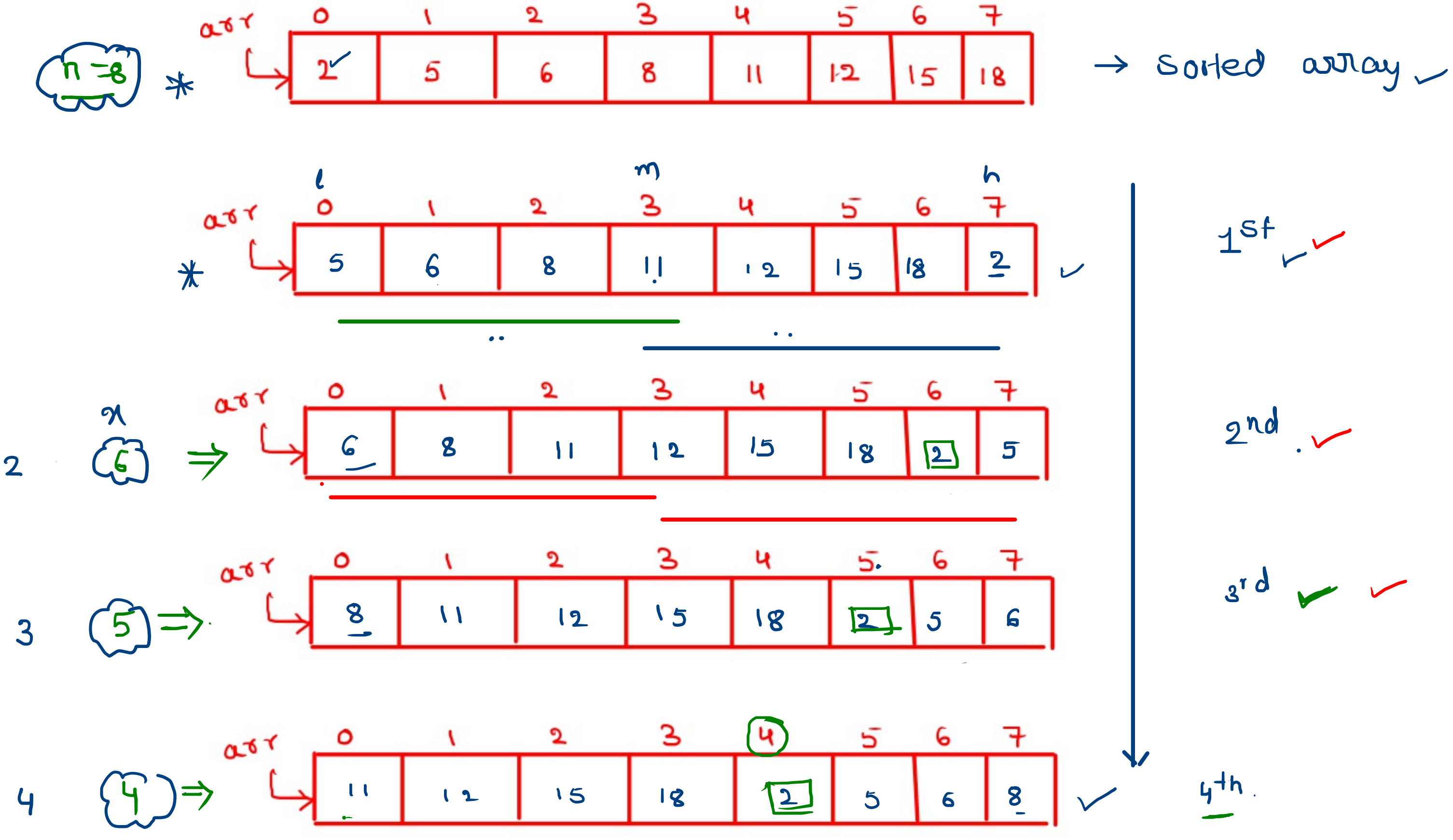
Day-11

1) Find the number of times array is rotated [clock wise]

n=8



n =8 * arr

0	1	2	3	4	5	6	7
2	5	6	8	11	12	15	18

→ sorted array

* arr

0 (l)	1	2	3 (m)	4	5	6	7 (h)
5	6	8	11	12	15	18	2

1st

2 x 6 ⇒ arr

0	1	2	3	4	5	6	7
6	8	11	12	15	18	2	5

2nd

3 5 ⇒ arr

0	1	2	3	4	5	6	7
8	11	12	15	18	2	5	6

3rd

4 4 ⇒ arr

0	1	2	3	4	5	6	7
11	12	15	18	2	5	6	8

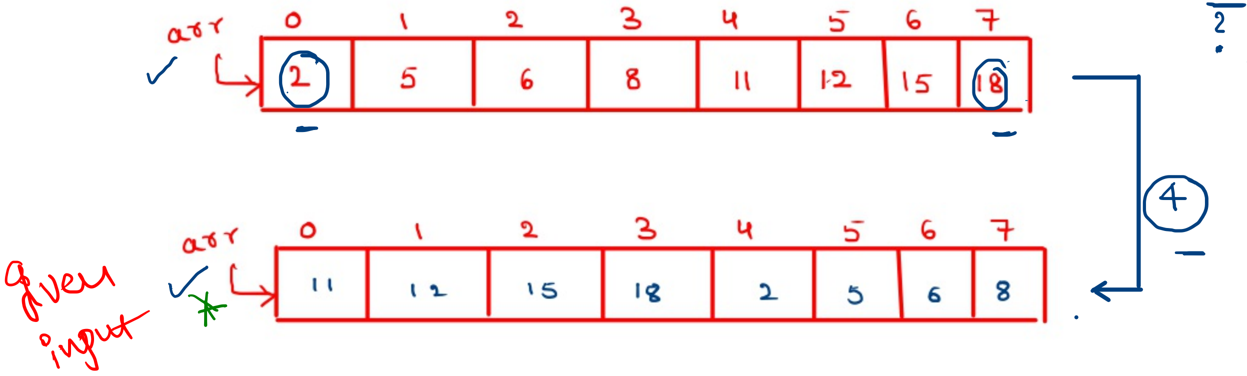
4th

n=8

x-times

2

input : rotated sorted array.



0	1	2	3	4	5	6	7
2	5	6	8	11	12	15	18

0	1	2	3	4	5	6	7
11	12	15	18	2	5	6	8

→ find the index of smaller element in the given array.

low high

⇒ arr ↳

0	1	2	3	4	5	6	7
2	5	6	8	11	12	15	18

* Sorted *

→ index (return)

Not Value.

key

low mid high

L R

⇒ arr ↳

0	1	2	3	4	5	6	7
11	12	15	18	2	5	6	8
l *	✓		m *		✓		h

L m

11 12 15 18

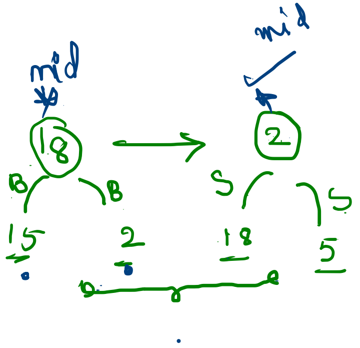
①

m h

18 2 5 6 8

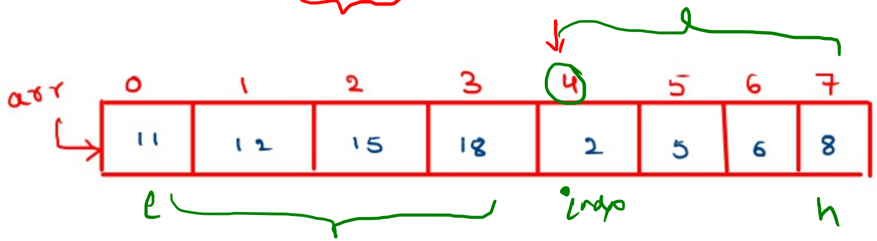
②

↳ un-sorted



```
function findMin(arr[],n) // find the number of times rotated
{
      low=0
      high=n-1
      if(arr[low]<=arr[high])     } → arr not at all rotated
               return 0
      while(low<=high)
      {
         → mid=low+(high-low)/2
            if(arr[mid])>arr[mid+1])          }
                     return n-mid+1           } mid
            else if(arr[mid]<arr[mid-1])      }
                     retur n-mid
            else if(arr[low]<=arr[mid])//R.H.S unsorted region
                     low=mid+1 ✓
            else if(arr[mid]<=arr[high])//LH.S unsorted region
                   high=mid-1 ✓
        }
     return -1; // to make compiler happy
}
```

* 2) Find an element in sorted rotated array



Key = 6 25 → -1

```
1) index=findMin(arr,n) →
      4
                              3
-1  2) x=BinarySearch(arr,0,index-1)
                              4      7
-1  3) y=BinarySearch(arr,index,n-1)

4) if(x==-1 && y==-1)
        return -1; //element is not present
5) if(x>=0)
        return x
   else
        return y
```

[11 12 15 18] → -1

[2 5 6 8] → 6

* 3) Search in a "nearly sorted array" [element that should suppose to present at ith location can present on (i-1)th location or ith location or (i+1)th location]

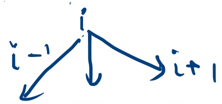
i/p arr [] = { 5, 10, 30, 20, 40 }, key =30 → 2

→ arr[ ]* = { 10, 30, 20, 40, 5 }

arr ↳

0	1	2	3	4
5	10	*30	20	40

5 10 20 30 40

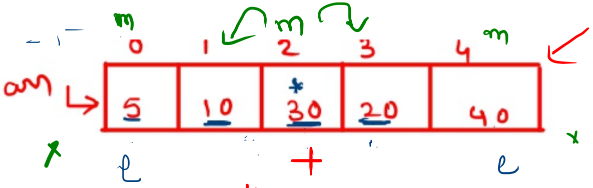


arr ↳

0	1	2	3	4
5	10	20	30	40

→ final sorted

* B·S



Key = 30

Nearly sorted

i → i →
i+1 →
i-1 →

Modified = B·S

→ arr[mid] v/s key
== } →

{
→ if (arr[mid] == key)
return mid

→ else if(mid-1 >= low && arr[mid-1] == key)
return mid-1

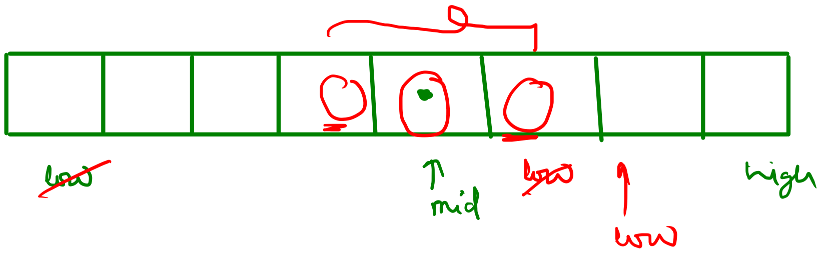
→ else if (mid+1 ≤ high && arr[mid+1] == key)
return mid+1
}

→ arr[mid] < key // RHS
low = mid+1 } →

{ → if(key > arr[mid]) // R.H.S
low = mid+2 ✓

→ key < arr[mid] || LHS
high = mid-1 } →

{ → if(key < arr[mid]) || LHS
high = mid-2 ✓
}

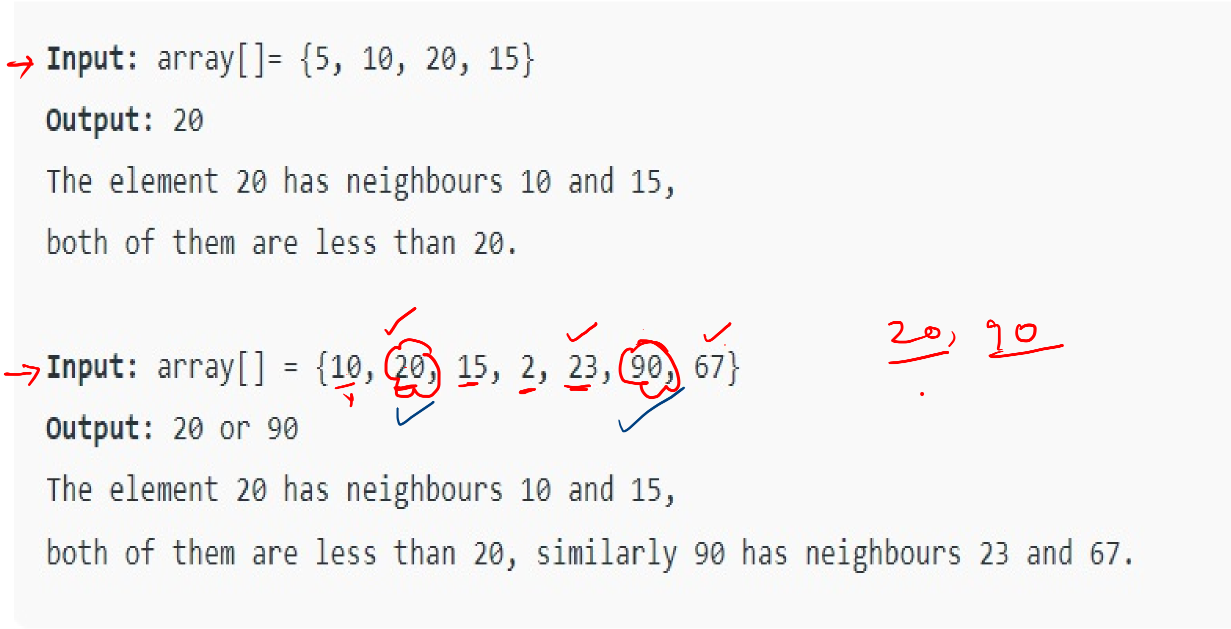


key = 70

4) Find the peak element in array

arr

0	1	2	3	4	5	6
10	20	15	2	23	90	67



Input: array[]= {5, 10, 20, 15}

Output: 20

The element 20 has neighbours 10 and 15,

both of them are less than 20.

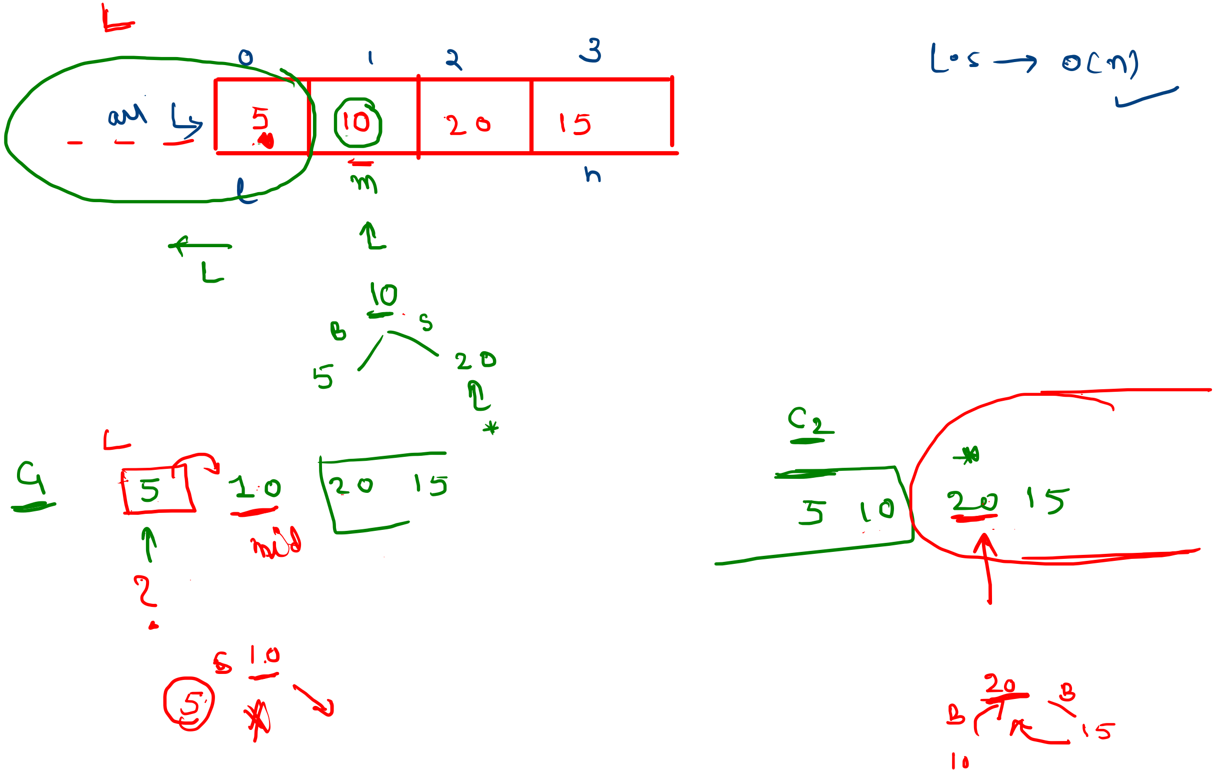
Input: array[] = {10, 20, 15, 2, 23, 90, 67}

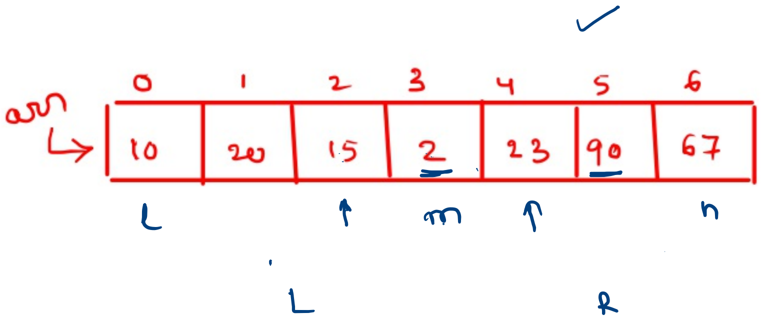
Output: 20 or 90

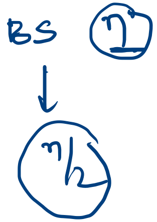
The element 20 has neighbours 10 and 15,

both of them are less than 20, similarly 90 has neighbours 23 and 67.

20, 90


L
0
1
2
3
arr
5
10
20
15
l
m
h
L
L
L.s → O(n)
10
B
S
5
20
L
*
C1
L
5
10
mid
20
15
2
S
10
5
*
C2
*
5
10
20
15
20
B
B
10
15


✓
0
1
2
3
4
5
6
arr
10
20
15
2
23
90
67
L
m
n
L
R


BS
n
n/2

```
function findPeak(arr[],n)
{
    low=0
    high=n-1
    while(low<=high)
    {
         mid=low+(high-low)/2
         if(mid>0 && mid<n-1) // skipping 1st and last element
           {
            if(arr[mid]>arr[mid+1] && arr[mid]>arr[mid-1])
                  return arr[mid]
            else if(arr[mid+1]>arr[mid])//R.H.S
                        low=mid+1
                 else
                        high=mid-1
           }
        else if(mid==0)
           {
                       if(arr[0]>arr[1]) return arr[0]
                    else  return arr[1]
          }
        else if(midd=n-1)
          {
                   if(arr[n-1]>arr[n-2]) return arr[n-1]
                   else           return arr[n-2]
         }
    }
   return -1;
}
```