Binary Search-2

✓
L.S ← arr, n, key

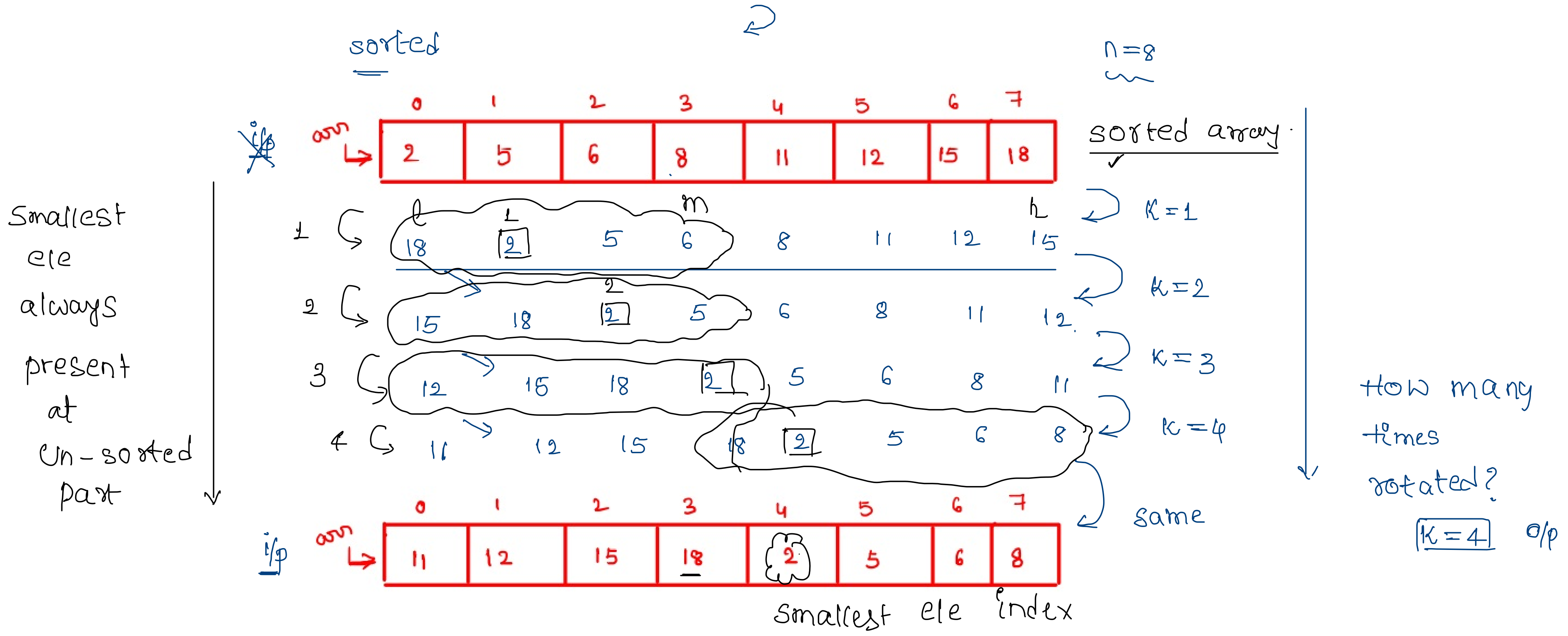
✓ ✓ ✓ ✓ -1

$O(n)$ ✓

BS : 1) sorted array + 2) Key $\}$ $O(\log_2^n)$

-1

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



number of times array rotated in cw direction = index number of the smallest element in rotated array arr →

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

$n=8$ ✓

sorted array.

~~8 times~~
~~16 tim~~
~~24~~

rotated
11 times

new array (i/p)

arr →

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

arr →

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

← possible ✓

ans: 0 ✓

~~mod~~ n

17 times = 1 time

$\cancel{8} + \cancel{8} + 1 = 17$ ✓

$n=8$

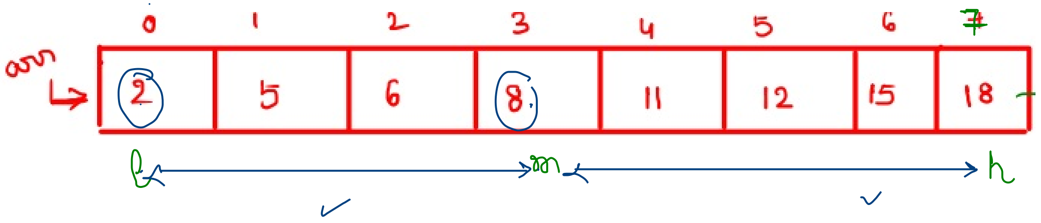
37 times $= 4(8) + 5$ ✓

0 - - - - 7

ans.

case1:- Array is sorted [i.e rotated 0, n, 2n, 3n .... times]

i/p :- purely sorted array.



```
if(arr[low]<=arr[mid] && arr[mid]<=arr[high])
        return low;
```

=> 0 times ✓

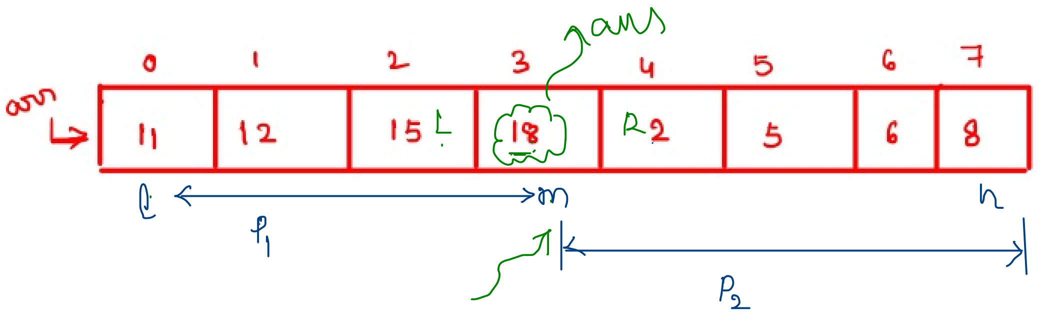
case2: Array is rotated :-

K=4

n=8

sorted arr
↓ rotated.

Smallest
ele
always
present
at
Un-sorted
part

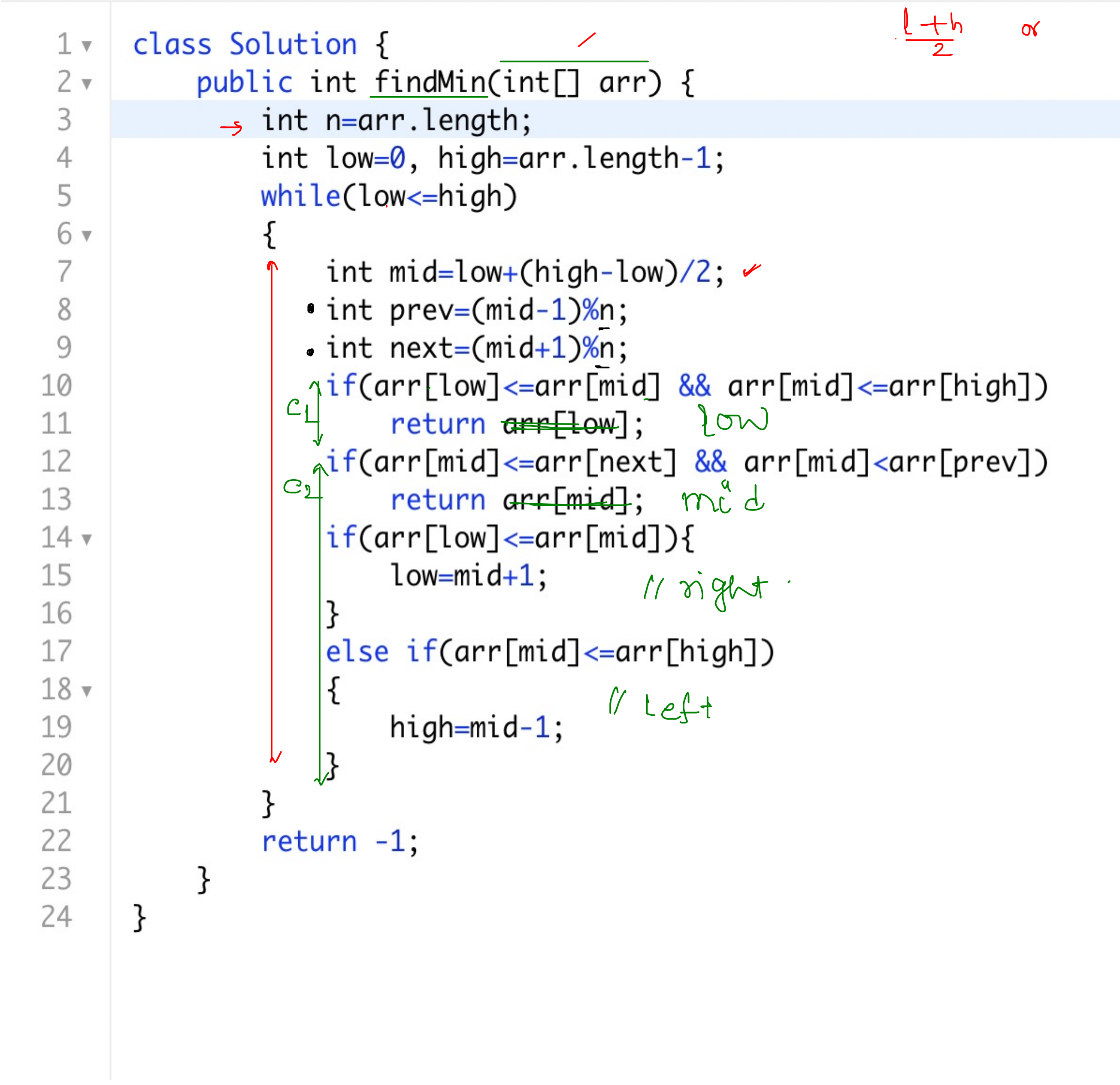


→ NOT sorted ?

one part is un-sorted

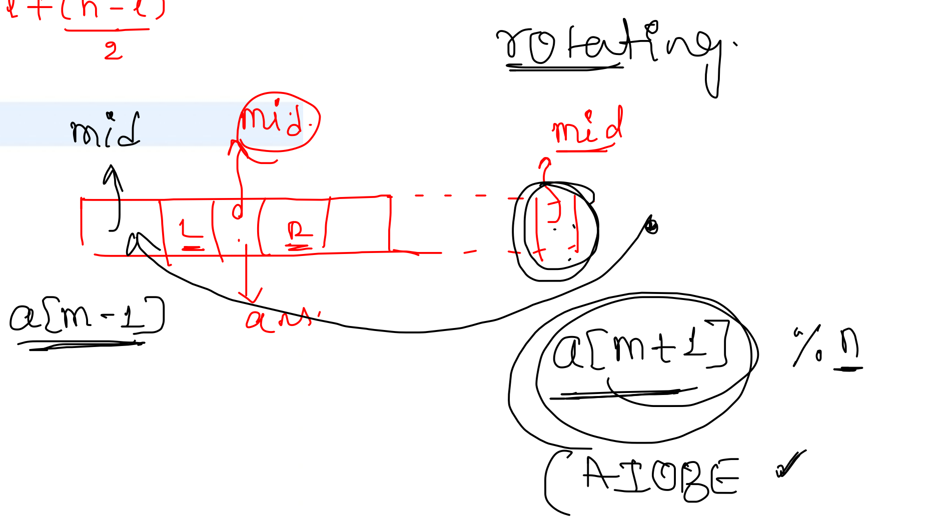
```
if(arr[mid]<=arr[mid+1] && arr[mid]<=arr[mid-1]) // when you are lucky
            return mid;


// un lucky : go for unsorted part, because you need smallest element

if(arr[low]<=arr[mid] )// low to mid is sorted, so I should go for right
{
            low=mid+1
}

if(arr[mid]<=arr[high]) // mid to high is sorted, so I should gor for left
{
            high=mid-1;
}
```

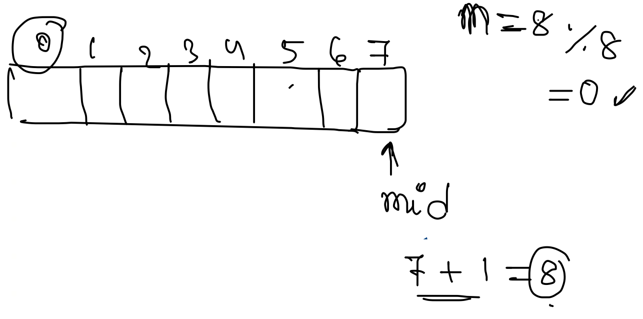
of times array rotated.



$\frac{l+h}{2}$ or ✓ preferable $l+\frac{(h-l)}{2}$

```
class Solution {
    public int findMin(int[] arr) {
      → int n=arr.length;
        int low=0, high=arr.length-1;
        while(low<=high)
        {
            int mid=low+(high-low)/2; ✓
          • int prev=(mid-1)%n;
          • int next=(mid+1)%n;
       c1 if(arr[low]<=arr[mid] && arr[mid]<=arr[high])
                return ~~arr[low]~~;   low
       c2 if(arr[mid]<=arr[next] && arr[mid]<arr[prev])
                return ~~arr[mid]~~;   mid
            if(arr[low]<=arr[mid]){
                low=mid+1;            // right
            }
            else if(arr[mid]<=arr[high])
            {
                high=mid-1;           // Left
            }
        }
        return -1;
    }
}
```





0, 8, 16, 32, ... (multiple of n)

② Search an element in a sorted and rotated array

(i/p)

arr ↳

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

n=8

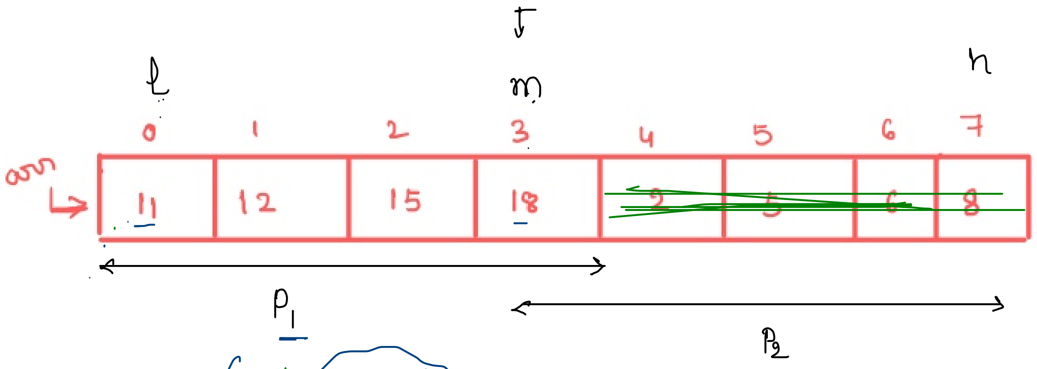
key = 12

o/p
↳ 1

BS → $O(\log_2 n)$
① array must be sorted.
② key

case1:- Array is completely sorted

```
✓if(arr[low]<=arr[mid] && arr[mid]<=arr[high])
 {
          return binarySearch(arr,low,high,key);
 }
```



n=8

Key = ~~12~~ 6 ✓

1 n=1000

✓ 1 which part is sorted

+

2 ~~Blindly go there~~ ✓

key is present/Not

inthat part check

↳ sorted, key } BS ✓

~~key~~ $\geq$ a[low] && key $\leq$ a[mid]

2nd region

1 2 3 4 100

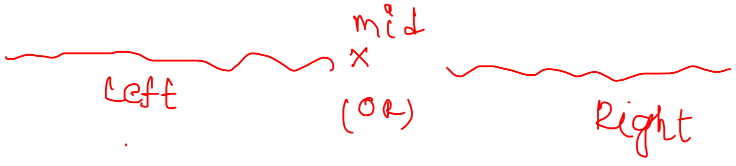
Key = ~~91~~ 101

```
function  search(arr[],n,key)
{


}
```

arr

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

```
int findElement(int arr[], int n, int target)
{
          int low=0, high=n-1;
          while(low<=high)
          {
             ✓ int mid=low+(high-low)/2;
               if(arr[mid]==target) } Lucky
                     return mid;    }
          C1 else if(arr[low]<=arr[mid]) // low to mid is sorted ✓
               {
                    if(target>=arr[low] && target<arr[mid])  →
                               high=mid-1 ✓ Left Half
                         else
                              low=mid+1 ✓ Right Half
               }
          C2 else if(arr[mid]<=arr[high]) // mid to high is sorted
             {
                   if(target>arr[mid] && target <=arr[high])
                        low=mid+1 // Right
                      else
                           high=mid-1 // Left
               }
          }
          return -1 ✓
}
```



① sortedness ✓ ~~Blindly~~

target ✓