Find Last Occurrence

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    int target = scn.nextInt();
    System.out.println(BSUB(arr, n, target));
}
public static int BSUB(int[] arr, int n, int target) {
    int i = 0;
    int j = n - 1;
    while ( i <= j ) {
        int mid = (i + j) / 2;
        if ( target == arr[mid] ) {
            if ( mid + 1 < n && arr[mid] == arr[mid + 1] ) {
                i = mid + 1;
            } else {
                return mid;
            }
        } else if ( target < arr[mid] ) {
            j = mid - 1;
        } else {
            i = mid + 1;
        }
    }
    return -1;
}
```

T.C = O(logN)

S.C = O(1)

Find The Index of Rotation (distinct)

arr = [8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7]

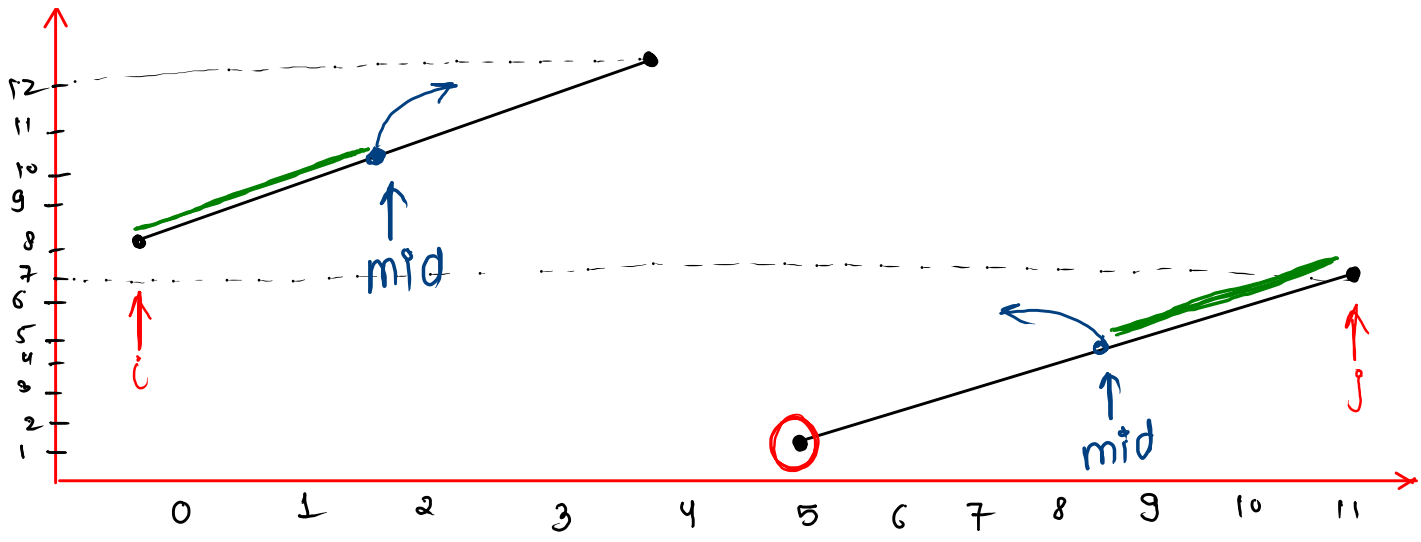
0 1 2 3 4 5 6 7 8 9 10 11

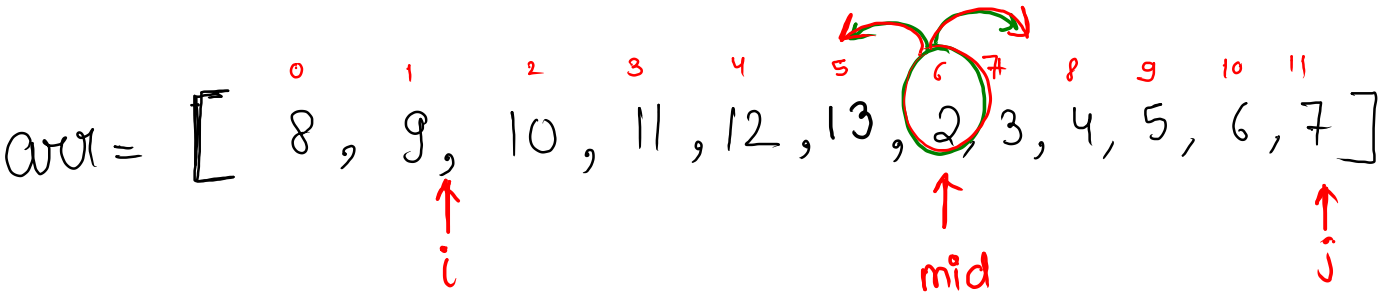
i, mid, mid, j

ans = 4

target = smallest element idx

my goal






```
i = 0, j = n-1;
while ( i<=j)
    mid = (i+j)/2;
    next = (mid+1) % n;
    prev = (mid-1+n) % n;
  if (arr[mid] <= arr[prev] && arr[mid] <= arr[next] ) {
a [     return mid-1;
  } else if ( arr[mid] <= arr[j]){
b [     j = mid-1;
  } else if (arr[mid] >= arr[i] ) {
c [     i = mid+1;
  }
}
```

Note :-

size

clockwise rotation = $(x \% n)$

anti-clockwise rotation = $(x + n) \% n$

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    System.out.println(findIndex(arr, n));
}
public static int findIndex(int[] arr, int n) {
    int i = 0;
    int j = n - 1;
    while ( i <= j ) {
        int mid = (i + j) / 2;
        int prev = (mid - 1 + n) % n;
        int next = (mid + 1) % n;
        if ( arr[mid] <= arr[prev] && arr[mid] <= arr[next] ) {
            return mid - 1;
        } else if ( arr[mid] <= arr[j] ) {
            j = mid - 1;
        } else if ( arr[mid] >= arr[i] ) {
            i = mid + 1;
        }
    }
    return -1;
}
```

Find Square Root

(Note:- Binary Search can be applied on an imaginary range as well)

n = 64 , n = 79

ans = 8 ans = 8.76

= 8

---

n = 30

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 30

i = 1

j = 30

mid = ~~15~~ ~~7~~ ~~3~~ ~~5~~ 6

ans = j

mid * mid == n

mid * mid < n , i = mid + 1 ;

mid * mid > n , j = mid - 1 ;

dry run

n = 48

mid

1 ..... 6 7 8 9 11 12 ..... 23 24 48

j i

```
i=1, j=n;
while (i<=j) {
        int mid = (i+j)/2;
        if (mid*mid == n){
a           return mid;
        } else if (mid*mid < n) {
b           i = mid+1;
        } else if (mid*mid > n) {
c           j = mid-1;
        }
}
```

T.C = O(logN)

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    System.out.println(findSqrt(n));
}
public static int findSqrt(int n) {
    int i = 1;
    int j = n;
    while ( i <= j ) {
        int mid = (i + j) / 2;
        if ( mid * mid == n ) {
            return mid;
        } else if ( mid * mid < n ) {
            i = mid + 1;
        } else if ( mid * mid > n ) {
            j = mid - 1;
        }
    }
    return j;
}
```

T.C = O(log N)

S.C = O(1)

use j as an answer when looking for just smaller element.