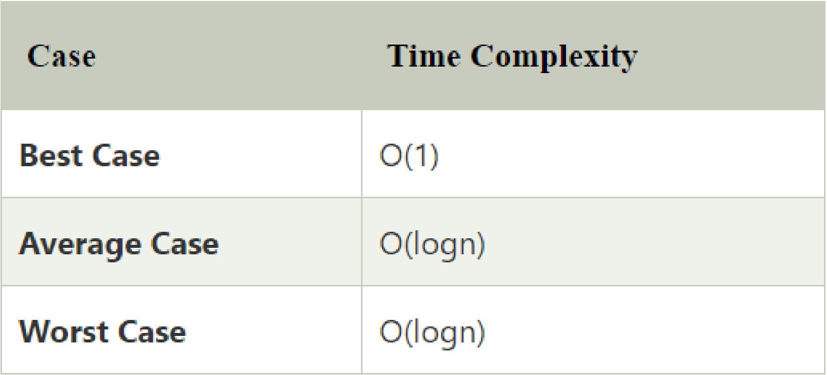
Binary Search

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

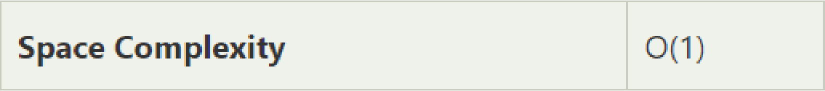
Binary Search Algorithm: The basic steps to perform Binary Search are:

- Sort the array in ascending order.
- Set the low index to the first element of the array and the high index to the last element.
- Set the middle index to the average of the low and high indices.
- If the element at the middle index is the target element, return the middle index.
- If the target element is less than the element at the middle index, set the high index to the middle index – 1.
- If the target element is greater than the element at the middle index, set the low index to the middle index + 1.
- Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array.

Aniket Meshram

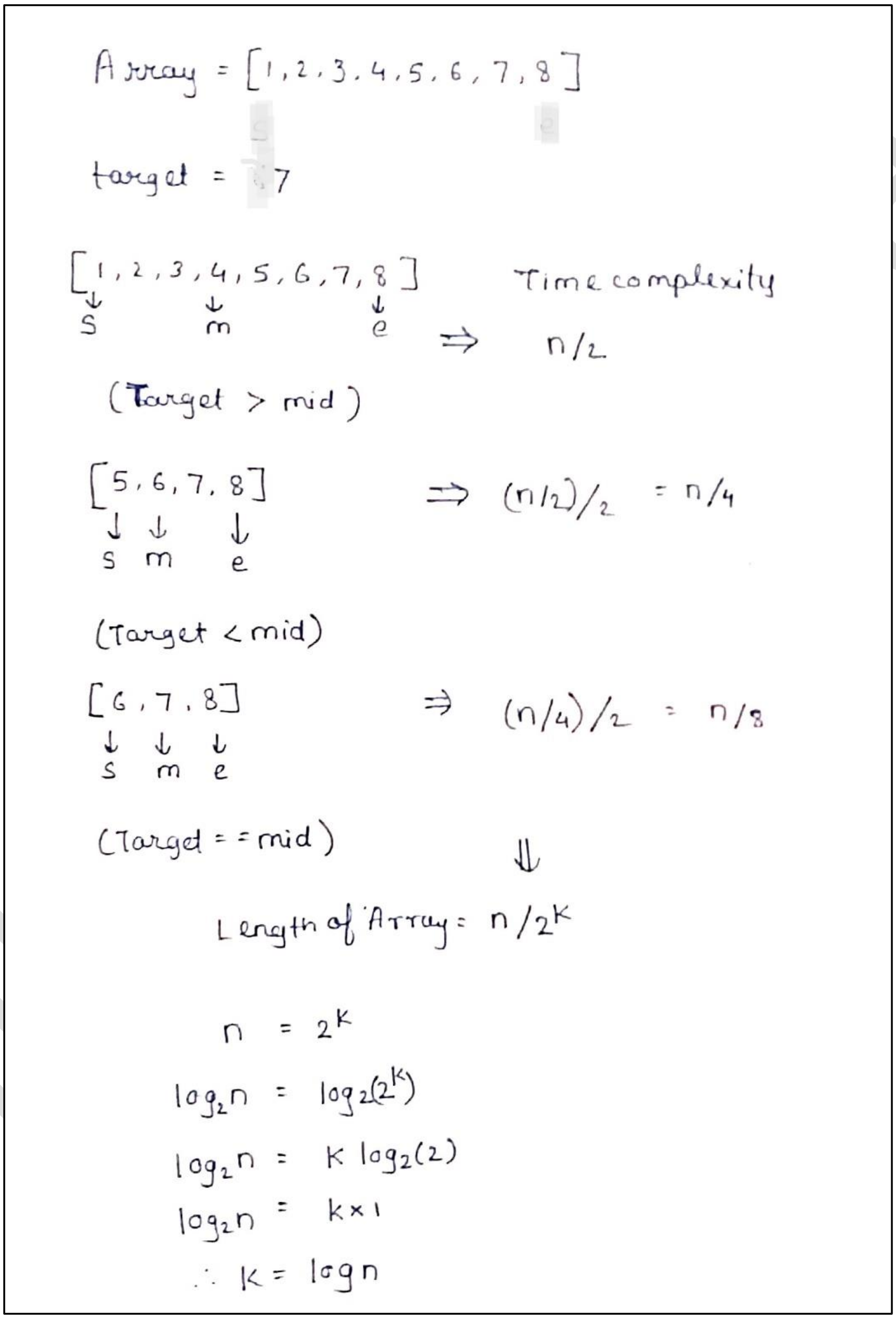


Case	Time Complexity
Best Case	O(1)
Average Case	O(logn)
Worst Case	O(logn)



Space Complexity	O(1)

Time complexity



Array = [1, 2, 3, 4, 5, 6, 7, 8]

target = 7

[1, 2, 3, 4, 5, 6, 7, 8]

$\overset{\downarrow}{s}$ $\overset{\downarrow}{m}$ $\overset{\downarrow}{e}$

Time complexity

$\Rightarrow$ $n/2$

(Target > mid)

[5, 6, 7, 8]

$\overset{\downarrow}{s}$ $\overset{\downarrow}{m}$ $\overset{\downarrow}{e}$

$\Rightarrow$ $(n/2)/2 = n/4$

(Target < mid)

[6, 7, 8]

$\overset{\downarrow}{s}$ $\overset{\downarrow}{m}$ $\overset{\downarrow}{e}$

$\Rightarrow$ $(n/4)/2 = n/8$

(Target = = mid)

$\Downarrow$

Length of Array = $n/2^K$

$$n = 2^K$$

$$\log_2 n = \log_2(2^K)$$

$$\log_2 n = K \log_2(2)$$

$$\log_2 n = K \times 1$$

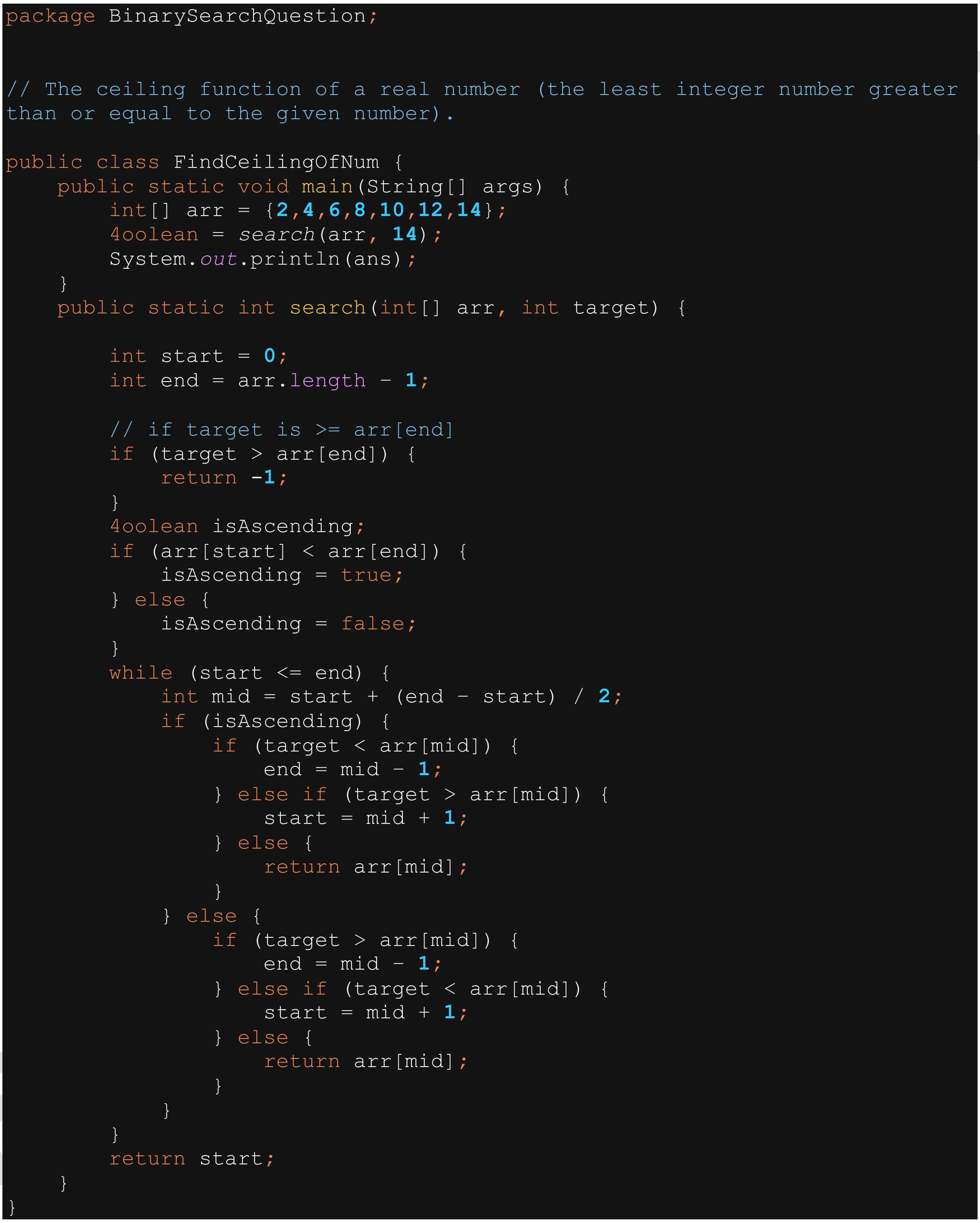
$$\therefore K = \log n$$

Binary Search Program

```
package BinarySearch;

public class BinarySearch {
    public static void main(String[] args) {
        int[] arr = {233,212,32,23,11,3,1,1};
        int ans = search(arr, 1);
        System.out.println(ans);
    }


    public static int search(int[] arr, int target) {
        int start = 0;
        int end = arr.length-1;



        // check weather array is sorted in ascending or descending order
        boolean isAscending;

        if (arr[start] < arr[end]) {
            isAscending = true;
        }
        else {
            isAscending = false;
        }



        while (start <= end) {
            // value of start and end is updating inside this block that's
why we make mid variable inside the block
            int mid = (end + start) / 2;

            // if order is ascending then do this
            if (isAscending) {
                if (target < arr[mid]) {
                    end = mid - 1;
                } else if (target > arr[mid]) {
                    start = mid + 1;
                } else {
                    return mid;
                }
            }

            // if the order is descending then do this
             else  {
                if (target > arr[mid]) {
                    end = mid - 1;
                } else if (target < arr[mid]) {
                    start = mid + 1;
                } else {
                    return mid;
                }
            }
        }

        // if element not found then print -1;
        return -1;
    }
```

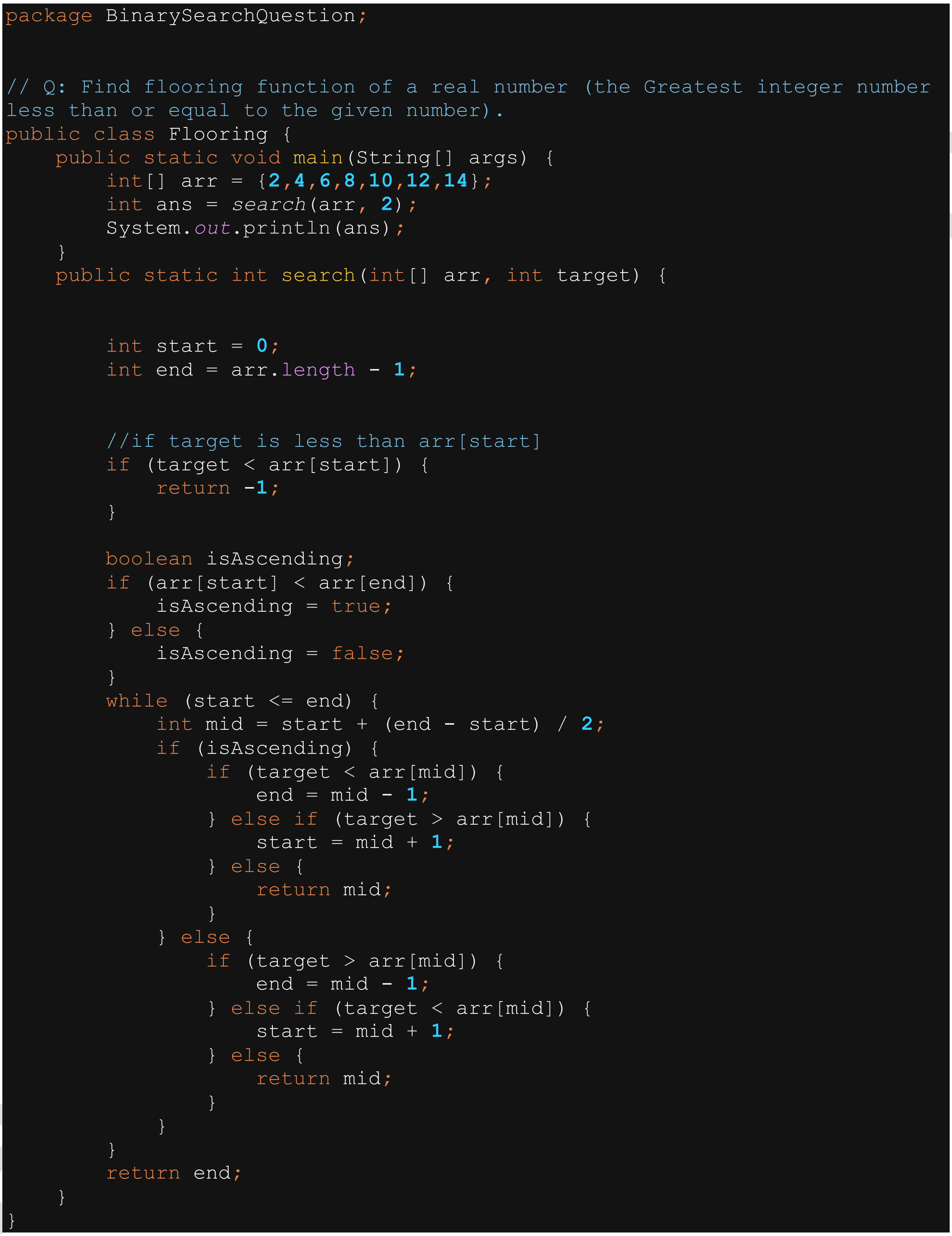
Questions on Binary Search

Find Ceiling of a Natural Number (smallest number greater than equal to target)



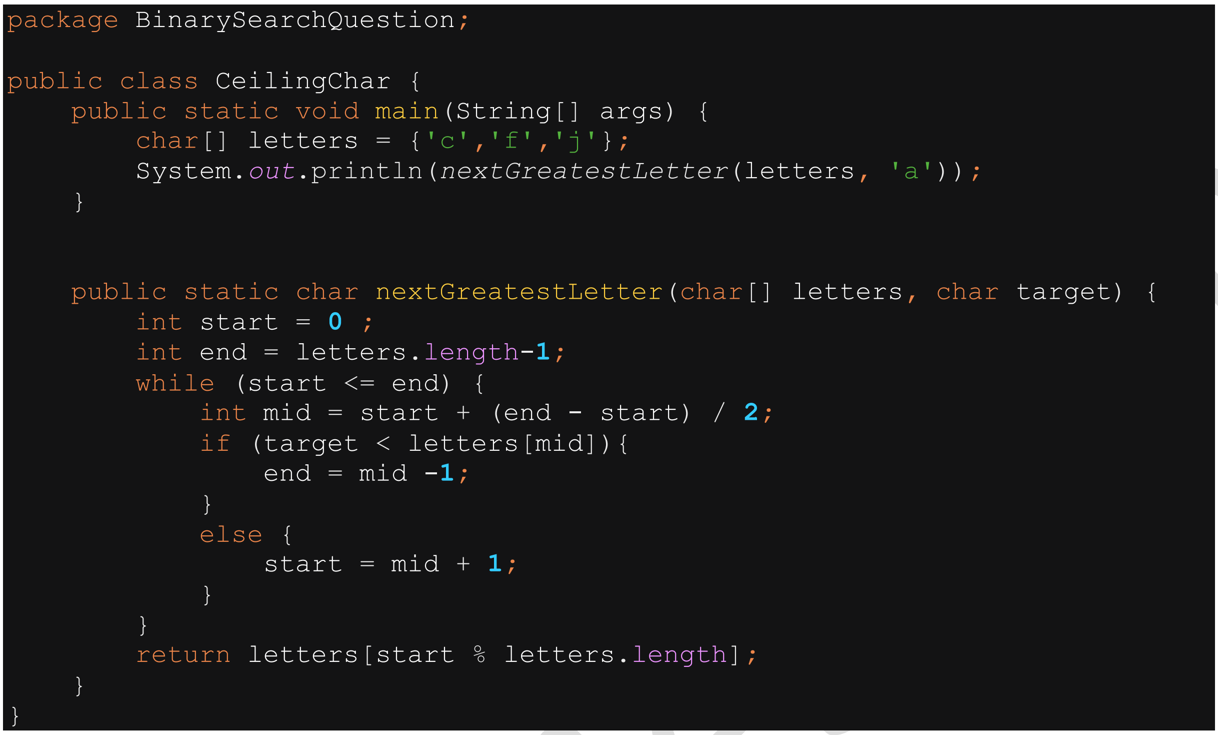
```java
package BinarySearchQuestion;


// The ceiling function of a real number (the least integer number greater
than or equal to the given number).

public class FindCeilingOfNum {
    public static void main(String[] args) {
        int[] arr = {2,4,6,8,10,12,14};
        4oolean = search(arr, 14);
        System.out.println(ans);
    }
    public static int search(int[] arr, int target) {

        int start = 0;
        int end = arr.length - 1;

        // if target is >= arr[end]
        if (target > arr[end]) {
            return -1;
        }
        4oolean isAscending;
        if (arr[start] < arr[end]) {
            isAscending = true;
        } else {
            isAscending = false;
        }
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (isAscending) {
                if (target < arr[mid]) {
                    end = mid - 1;
                } else if (target > arr[mid]) {
                    start = mid + 1;
                } else {
                    return arr[mid];
                }
            } else {
                if (target > arr[mid]) {
                    end = mid - 1;
                } else if (target < arr[mid]) {
                    start = mid + 1;
                } else {
                    return arr[mid];
                }
            }
        }
        return start;
    }
}
```

Find flooring of a real number (greatest number less than equal to target)



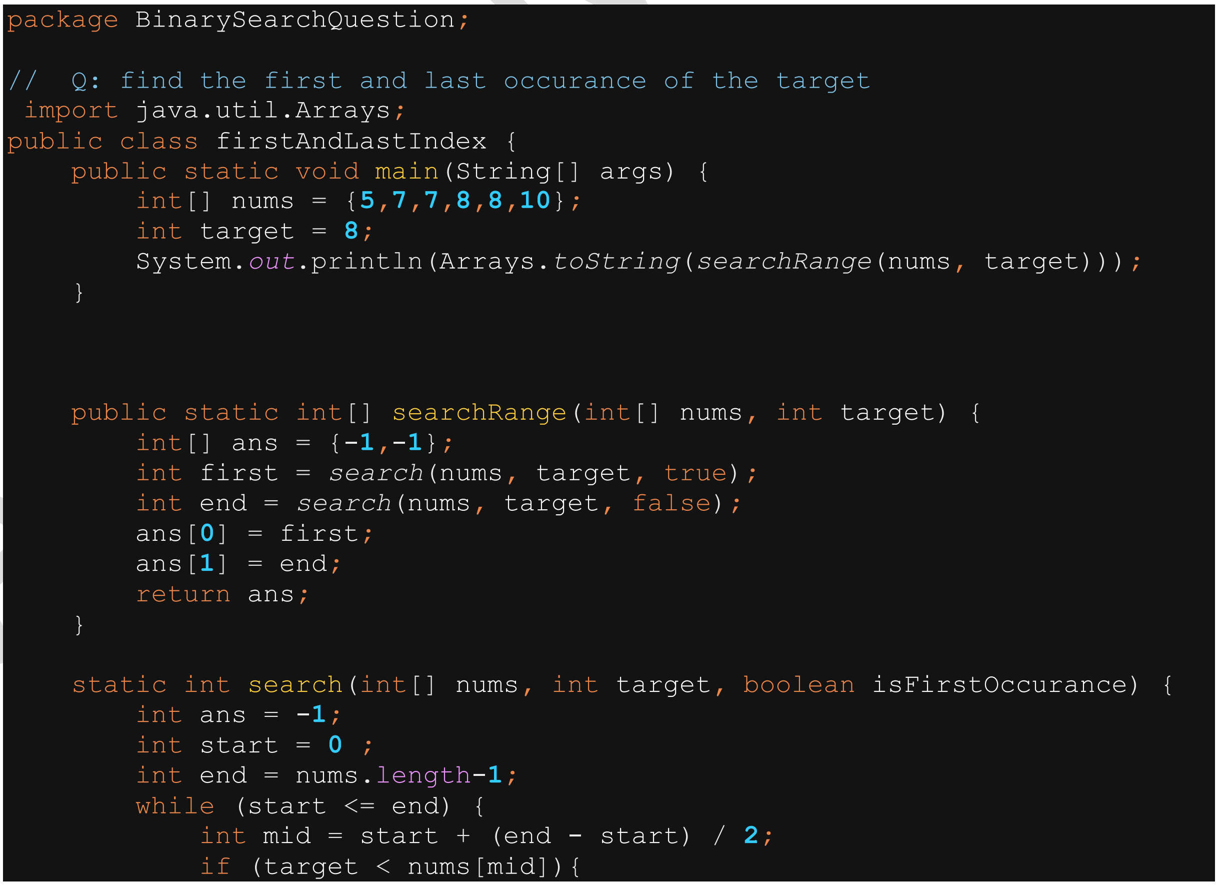
```java
package BinarySearchQuestion;


// Q: Find flooring function of a real number (the Greatest integer number
less than or equal to the given number).
public class Flooring {
    public static void main(String[] args) {
        int[] arr = {2,4,6,8,10,12,14};
        int ans = search(arr, 2);
        System.out.println(ans);
    }
    public static int search(int[] arr, int target) {


        int start = 0;
        int end = arr.length - 1;


        //if target is less than arr[start]
        if (target < arr[start]) {
            return -1;
        }

        boolean isAscending;
        if (arr[start] < arr[end]) {
            isAscending = true;
        } else {
            isAscending = false;
        }
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (isAscending) {
                if (target < arr[mid]) {
                    end = mid - 1;
                } else if (target > arr[mid]) {
                    start = mid + 1;
                } else {
                    return mid;
                }
            } else {
                if (target > arr[mid]) {
                    end = mid - 1;
                } else if (target < arr[mid]) {
                    start = mid + 1;
                } else {
                    return mid;
                }
            }
        }
        return end;
    }
}
```

Find ceiling of given target (smallest letter greater than equal to target)



```
package BinarySearchQuestion;

public class CeilingChar {
    public static void main(String[] args) {
        char[] letters = {'c','f','j'};
        System.out.println(nextGreatestLetter(letters, 'a'));
    }


    public static char nextGreatestLetter(char[] letters, char target) {
        int start = 0 ;
        int end = letters.length-1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (target < letters[mid]){
                end = mid -1;
            }
            else {
                start = mid + 1;
            }
        }
        return letters[start % letters.length];
    }
}
```

Find the first and last occurance of target



```
package BinarySearchQuestion;

//  Q: find the first and last occurance of the target
 import java.util.Arrays;
public class firstAndLastIndex {
    public static void main(String[] args) {
        int[] nums = {5,7,7,8,8,10};
        int target = 8;
        System.out.println(Arrays.toString(searchRange(nums, target)));
    }



    public static int[] searchRange(int[] nums, int target) {
        int[] ans = {-1,-1};
        int first = search(nums, target, true);
        int end = search(nums, target, false);
        ans[0] = first;
        ans[1] = end;
        return ans;
    }

    static int search(int[] nums, int target, boolean isFirstOccurance) {
        int ans = -1;
        int start = 0 ;
        int end = nums.length-1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (target < nums[mid]){
                end = mid -1;
            }
            else if (target > nums[mid]) {
                start = mid + 1;
            }
            else {
                ans = mid;
                if (isFirstOccurance) {
                    end = mid - 1;
                }
                else {
                    start = mid + 1;
                }
            }
        }
        return ans;

    }
}
```

Find the position of element of element in an infinite array

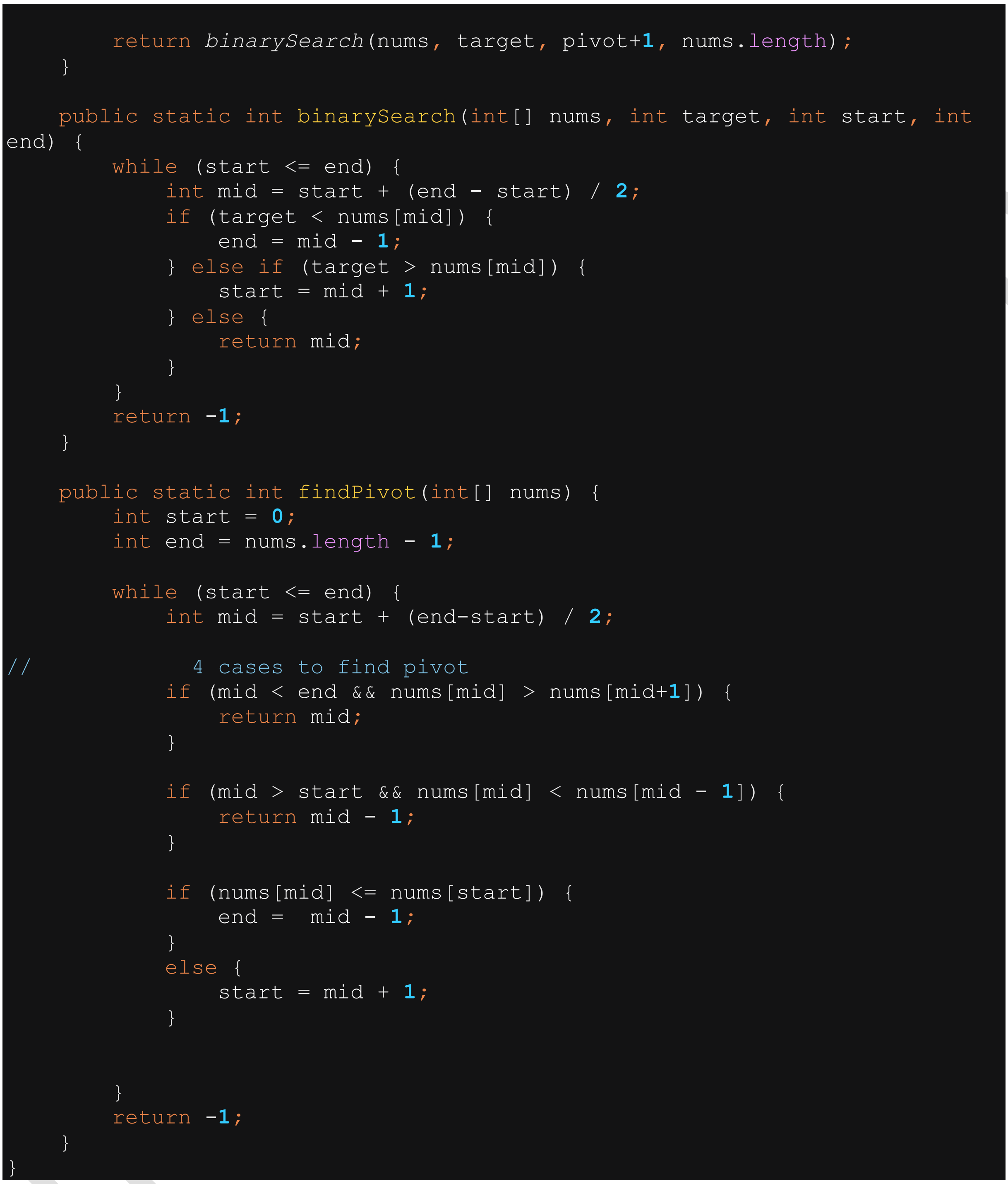
```
package BinarySearchQuestion;

//Q: position of element an element in an infinite array
public class PositionOfElement {
    public static void main(String[] args) {
        int[] arr = {4,13,15,116,123,200};
        System.out.print(ans(arr,15));
    }

    public static int ans(int[] arr, int target) {
        int start = 0;
        int end = 1;
        if (target > arr[end]) {
            int newStart = end + 1;
            end = end + (end - start + 1) * 2;
            start = newStart;
        }
        return Search(arr, target, start, end);
    }

    public static int Search(int[] arr, int target, int start, int end) {
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (target < arr[mid]) {
                end = mid - 1;
            }
            else if (target > arr[mid]) {
                start = mid + 1;
            }
            else{
                return mid;
            }
        }
        return -1;
    }
}
```

Find the peak index of mountain

```java
package BinarySearchQuestion;

//find the peak index of mountain

public class FindPeakIndex {
    public static void main(String[] args) {
        int[] arr = {4,4,5,6,7,3,2,1};
        System.out.println(search(arr));
    }

    public static int search (int[] arr) {
        int start = 0 ;
        int end = arr.length-1;
        while (start < end) {
            int mid = start + (end - start) / 2;
            if (arr[mid] < arr[mid + 1]) {
                start = mid + 1;
            }

            else {
                end = mid;
            }
        }
        return start;
    }
}
```

Search in Rotated Array

```java
package BinarySearchQuestion;


//  Q: Search in Rotated Sorted Array
public class SearchInRotatedArray {
    public static void main(String[] args) {
        int[] nums = {4,5,6,7,8,9,10,0,1,2};
        int target = 10;
        System.out.println(search(nums, target));
    }


    static int search(int[] nums, int target) {
        int pivot = findPivot(nums);


        //  if pivot is not found the do the normal binary search
        if (pivot == -1) {
           return binarySearch(nums, target, 0 , nums.length);
        }

        if (nums[pivot] == target){
            return pivot;
        }

        if (target >= nums[0]){
            return binarySearch(nums, target, 0, pivot-1);
        }
```



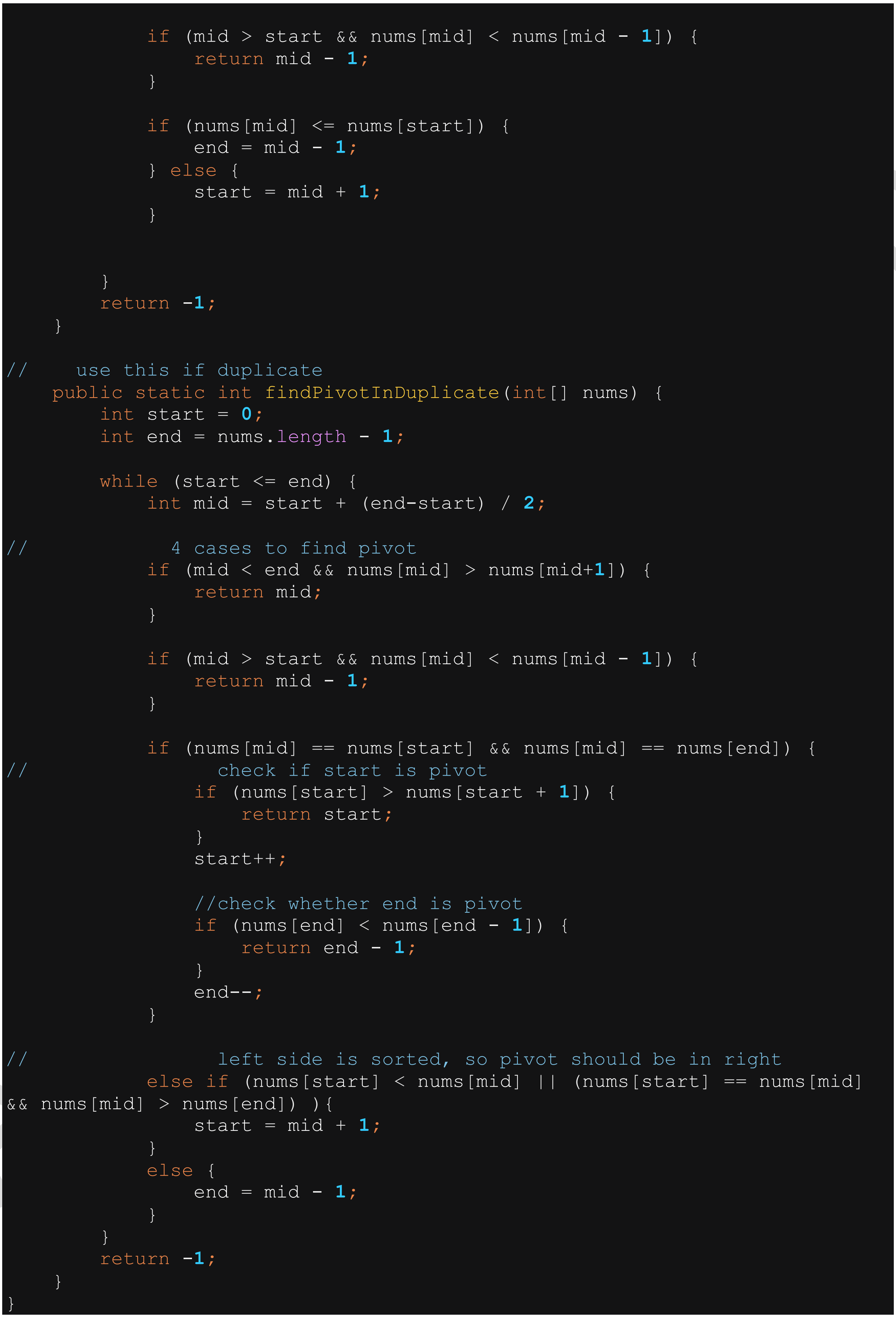
```
        return binarySearch(nums, target, pivot+1, nums.length);
    }

    public static int binarySearch(int[] nums, int target, int start, int
end) {
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (target < nums[mid]) {
                end = mid - 1;
            } else if (target > nums[mid]) {
                start = mid + 1;
            } else {
                return mid;
            }
        }
        return -1;
    }

    public static int findPivot(int[] nums) {
        int start = 0;
        int end = nums.length - 1;

        while (start <= end) {
            int mid = start + (end-start) / 2;

//              4 cases to find pivot
            if (mid < end && nums[mid] > nums[mid+1]) {
                return mid;
            }

            if (mid > start && nums[mid] < nums[mid - 1]) {
                return mid - 1;
            }

            if (nums[mid] <= nums[start]) {
                end =  mid - 1;
            }
            else {
                start = mid + 1;
            }


        }
        return -1;
    }
}
```

Search in duplicate rotated array

```
package BinarySearchQuestion;

//Q: Search In duplicate Rotated Array
public class SearchInDuplicateRotatedArray {
    public static void main(String[] args) {
        int[] nums = {2,2,3,3,4,0,0,1};
        int target = 3;
        System.out.println(search(nums, target));
    }


    static int search(int[] nums, int target) {
        int pivot = findPivotInDuplicate(nums);


        //  if pivot is not found the do the normal binary search
        if (pivot == -1) {
            return binarySearch(nums, target, 0 , nums.length);
        }

        if (nums[pivot] == target){
            return pivot;
        }

        if (target >= nums[0]){
            return binarySearch(nums, target, 0, pivot-1);
        }

        return binarySearch(nums, target, pivot+1, nums.length);
    }

    public static int binarySearch(int[] nums, int target, int start, int
end) {
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (target < nums[mid]) {
                end = mid - 1;
            } else if (target > nums[mid]) {
                start = mid + 1;
            } else {
                return mid;
            }
        }
        return -1;
    }

    public static int findPivotInDuplicate(int[] nums) {
        int start = 0;
        int end = nums.length - 1;

        while (start <= end) {
            int mid = start + (end-start) / 2;

//             4 cases to find pivot
            if (mid < end && nums[mid] > nums[mid+1]) {
                return mid;
            }

            if (mid > start && nums[mid] < nums[mid - 1]) {
                return mid - 1;
            }

            if (nums[mid] == nums[start] && nums[mid] == nums[end]) {
//                check if start is pivot
                if (nums[start] > nums[start + 1]) {
                    return start;
                }
                start++;

                //check whether end is pivot
                if (nums[end] < nums[end - 1]) {
                    return end - 1;
                }
                end--;
            }

//                left side is sorted, so pivot should be in right
            else if (nums[start] < nums[mid] || (nums[start] == nums[mid]
&& nums[mid] > nums[end]) ){
                start = mid + 1;
            }
            else {
                end = mid - 1;
            }
        }
        return -1;
    }
}
```

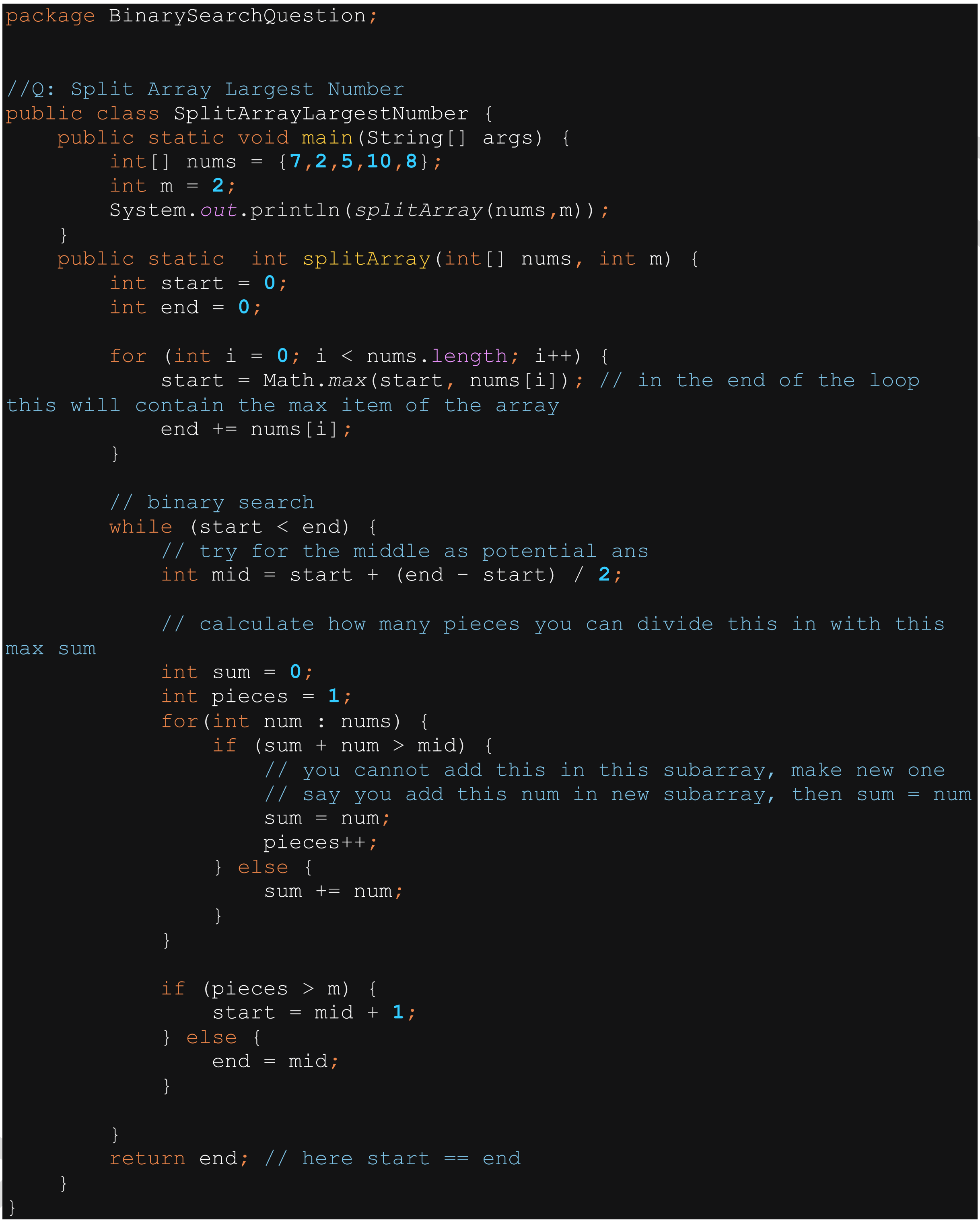
Rotation count

```
package BinarySearchQuestion;

public class RotationCount{
    public static void main(String[] args) {
        int[] nums = {3,4,5,0,1,2};
        System.out.println(count(nums));
    }

    public static int count(int[] nums) {
        int pivot = findPivot(nums);
        return pivot + 1;


    }

//    use this if non duplicate
    public static int findPivot(int[] nums) {
        int start = 0;
        int end = nums.length - 1;

        while (start <= end) {
            int mid = start + (end - start) / 2;

//            4 cases to find pivot
            if (mid < end && nums[mid] > nums[mid + 1]) {
                return mid;
            }
```



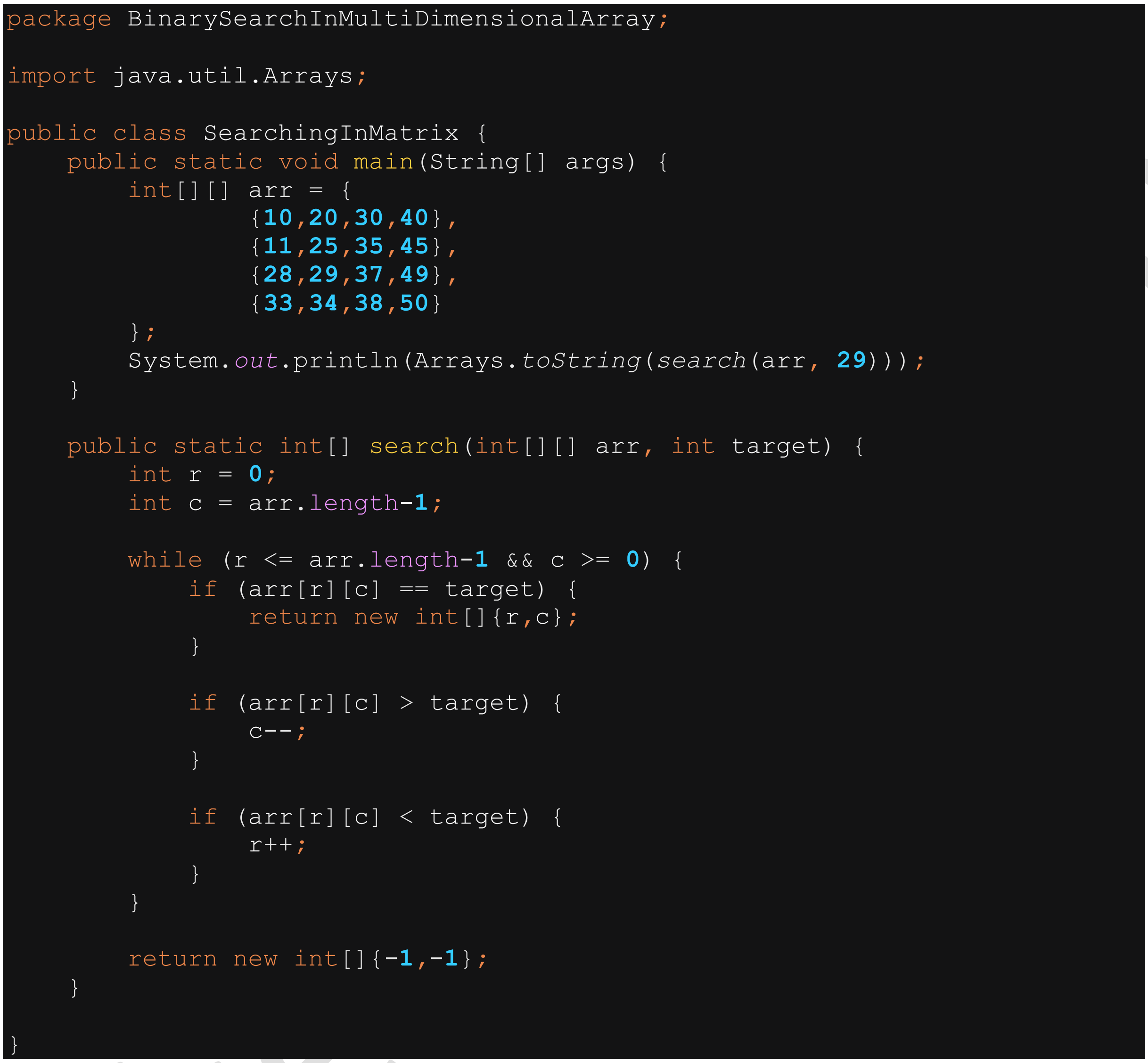
```
            if (mid > start && nums[mid] < nums[mid - 1]) {
                return mid - 1;
            }

            if (nums[mid] <= nums[start]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }


        }
        return -1;
    }

//    use this if duplicate
    public static int findPivotInDuplicate(int[] nums) {
        int start = 0;
        int end = nums.length - 1;

        while (start <= end) {
            int mid = start + (end-start) / 2;

//            4 cases to find pivot
            if (mid < end && nums[mid] > nums[mid+1]) {
                return mid;
            }

            if (mid > start && nums[mid] < nums[mid - 1]) {
                return mid - 1;
            }

            if (nums[mid] == nums[start] && nums[mid] == nums[end]) {
//                check if start is pivot
                if (nums[start] > nums[start + 1]) {
                    return start;
                }
                start++;

                //check whether end is pivot
                if (nums[end] < nums[end - 1]) {
                    return end - 1;
                }
                end--;
            }

//                left side is sorted, so pivot should be in right
            else if (nums[start] < nums[mid] || (nums[start] == nums[mid]
&& nums[mid] > nums[end]) ){
                start = mid + 1;
            }
            else {
                end = mid - 1;
            }
        }
        return -1;
    }
}
```

Split Array Largest Sum



```java
package BinarySearchQuestion;


//Q: Split Array Largest Number
public class SplitArrayLargestNumber {
    public static void main(String[] args) {
        int[] nums = {7,2,5,10,8};
        int m = 2;
        System.out.println(splitArray(nums,m));
    }
    public static  int splitArray(int[] nums, int m) {
        int start = 0;
        int end = 0;

        for (int i = 0; i < nums.length; i++) {
            start = Math.max(start, nums[i]); // in the end of the loop
this will contain the max item of the array
            end += nums[i];
        }

        // binary search
        while (start < end) {
            // try for the middle as potential ans
            int mid = start + (end - start) / 2;

            // calculate how many pieces you can divide this in with this
max sum
            int sum = 0;
            int pieces = 1;
            for(int num : nums) {
                if (sum + num > mid) {
                    // you cannot add this in this subarray, make new one
                    // say you add this num in new subarray, then sum = num
                    sum = num;
                    pieces++;
                } else {
                    sum += num;
                }
            }

            if (pieces > m) {
                start = mid + 1;
            } else {
                end = mid;
            }

        }
        return end; // here start == end
    }
}
```

Search in Matrix

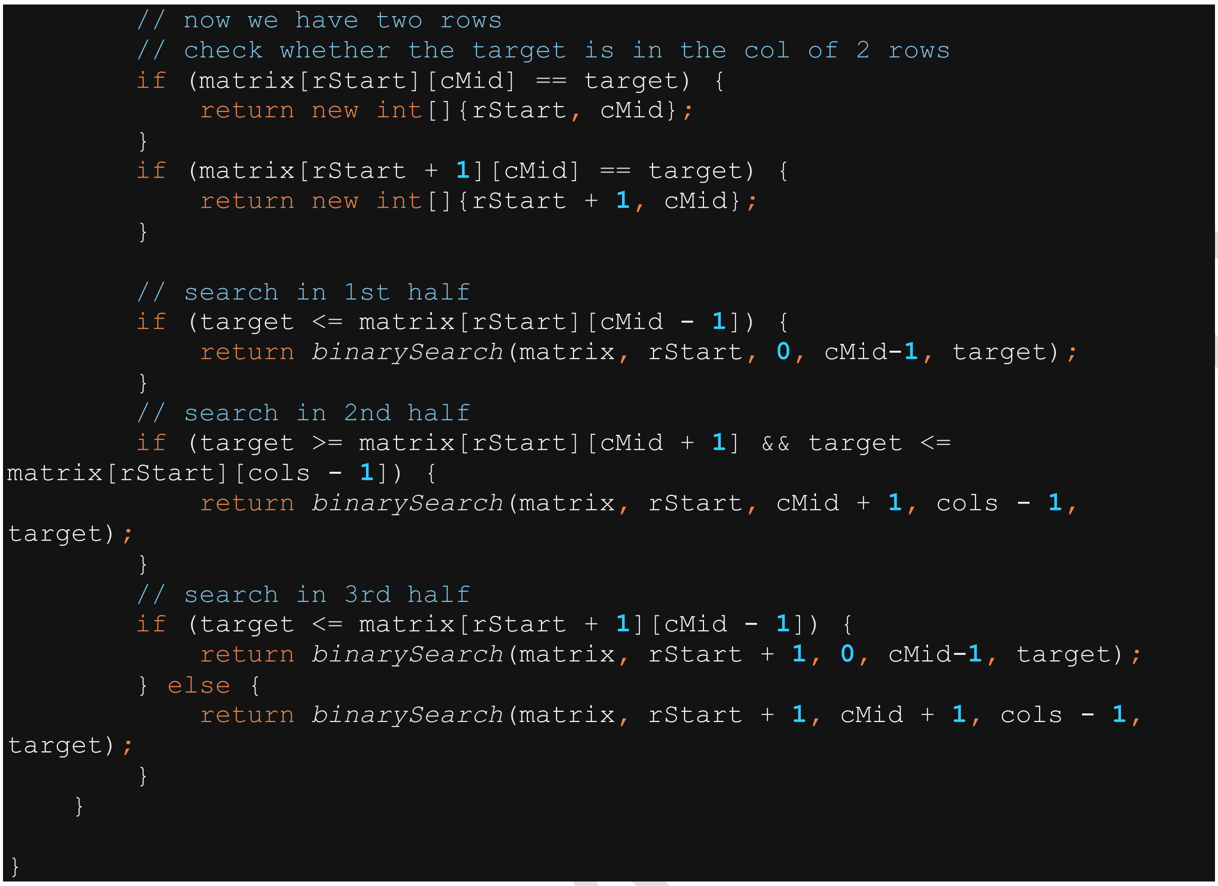


```
package BinarySearchInMultiDimensionalArray;

import java.util.Arrays;

public class SearchingInMatrix {
    public static void main(String[] args) {
        int[][] arr = {
                {10,20,30,40},
                {11,25,35,45},
                {28,29,37,49},
                {33,34,38,50}
        };
        System.out.println(Arrays.toString(search(arr, 29)));
    }

    public static int[] search(int[][] arr, int target) {
        int r = 0;
        int c = arr.length-1;

        while (r <= arr.length-1 && c >= 0) {
            if (arr[r][c] == target) {
                return new int[]{r,c};
            }

            if (arr[r][c] > target) {
                c--;
            }

            if (arr[r][c] < target) {
                r++;
            }
        }

        return new int[]{-1,-1};
    }

}
```

Anike

Search in Sorted matrix

```java
package BinarySearchInMultiDimensionalArray;


import java.util.Arrays;
public class SortedMatrix {
    public static void main(String[] args) {
        int[][] arr = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        System.out.println(Arrays.toString(search(arr, 9)));
    }

    // search in the row provided between the cols provided
    static int[] binarySearch(int[][] matrix, int row, int cStart, int
cEnd, int target) {
        while (cStart <= cEnd) {
            int mid = cStart + (cEnd - cStart) / 2;
            if (matrix[row][mid] == target) {
                return new int[]{row, mid};
            }
            if (matrix[row][mid] < target) {
                cStart = mid + 1;
            } else {
                cEnd = mid - 1;
            }
        }
        return new int[]{-1, -1};
    }

    static int[] search(int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length; // be cautious, matrix may be empty
        if (cols == 0){
            return new int[] {-1,-1};
        }
        if (rows == 1) {
            return binarySearch(matrix,0, 0, cols-1, target);
        }

        int rStart = 0;
        int rEnd = rows - 1;
        int cMid = cols / 2;

        // run the loop till 2 rows are remaining
        while (rStart < (rEnd - 1)) { // while this is true it will have
more than 2 rows
            int mid = rStart + (rEnd - rStart) / 2;
            if (matrix[mid][cMid] == target) {
                return new int[]{mid, cMid};
            }
            if (matrix[mid][cMid] < target) {
                rStart = mid;
            } else {
                rEnd = mid;
            }
        }
```



```
        // now we have two rows
        // check whether the target is in the col of 2 rows
        if (matrix[rStart][cMid] == target) {
            return new int[]{rStart, cMid};
        }
        if (matrix[rStart + 1][cMid] == target) {
            return new int[]{rStart + 1, cMid};
        }

        // search in 1st half
        if (target <= matrix[rStart][cMid - 1]) {
            return binarySearch(matrix, rStart, 0, cMid-1, target);
        }
        // search in 2nd half
        if (target >= matrix[rStart][cMid + 1] && target <=
matrix[rStart][cols - 1]) {
            return binarySearch(matrix, rStart, cMid + 1, cols - 1,
target);
        }
        // search in 3rd half
        if (target <= matrix[rStart + 1][cMid - 1]) {
            return binarySearch(matrix, rStart + 1, 0, cMid-1, target);
        } else {
            return binarySearch(matrix, rStart + 1, cMid + 1, cols - 1,
target);
        }
    }

}
```

Aniket M