Answer 1)

```
public class HelloWorld {
    public static int sqrt(int x) {
        if (x == 0 || x == 1) {
            return x;
        }

        int left = 1;
        int right = x;
        int result = 0;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (mid <= x / mid) {
                result = mid;
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int x1 = 4;
        System.out.println("Square root of " + x1 + ": " + sqrt(x1));

        int x2 = 16;
        System.out.println("Square root of " + x2 + ": " + sqrt(x2));

        int x3 = 10;
        System.out.println("Square root of " + x3 + ": " + sqrt(x3));
    }
}
```

Answer 2)

```
public class HelloWorld {
    public static int findPeakElement(int[] nums) {
        int left = 0;
```

```
        int right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] < nums[mid + 1]) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        return left;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 2, 3, 1};
        System.out.println("Peak element index: " + findPeakElement(nums1));

        int[] nums2 = {1, 2, 1, 3, 5, 6, 4};
        System.out.println("Peak element index: " + findPeakElement(nums2));
    }
}
```

Answer 3)

```
public class HelloWorld {
    public static int findMissingNumber(int[] nums) {
        int left = 0;
        int right = nums.length;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > mid) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
```

```
    public static void main(String[] args) {
        int[] nums1 = {3, 0, 1};
        System.out.println("Missing number: " + findMissingNumber(nums1));

        int[] nums2 = {0, 1};
        System.out.println("Missing number: " + findMissingNumber(nums2));

        int[] nums3 = {9, 6, 4, 2, 3, 5, 7, 0, 1};
        System.out.println("Missing number: " + findMissingNumber(nums3));
    }
}
```

Answer 4)

```
public class HelloWorld {
    public static int findRepeatedNumber(int[] nums) {
        int left = 1;
        int right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            int count = 0;
            for (int num : nums) {
                if (num <= mid) {
                    count++;
                }
            }

            if (count > mid) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 3, 4, 2, 2};
```

```
        System.out.println("Repeated number: " + findRepeatedNumber(nums1));

        int[] nums2 = {3, 1, 3, 4, 2};
        System.out.println("Repeated number: " + findRepeatedNumber(nums2));
    }
}
```

Answer 5)

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ArrayIntersection {
    public static int[] intersection(int[] nums1, int[] nums2) {
        // Sort the arrays
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        List<Integer> result = new ArrayList<>();

        int i = 0;
        int j = 0;

        // Find the intersection using binary search
        while (i < nums1.length && j < nums2.length) {
            if (nums1[i] == nums2[j]) {
                // Add the common element to the result
                if (result.isEmpty() || nums1[i] != result.get(result.size() - 1)) {
                    result.add(nums1[i]);
                }
                i++;
                j++;
            } else if (nums1[i] < nums2[j]) {
                i++;
            } else {
                j++;
            }
        }

        // Convert the result list to an array
        int[] intersection = new int[result.size()];
```

```
        for (int k = 0; k < result.size(); k++) {
            intersection[k] = result.get(k);
        }

        return intersection;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 2, 2, 1};
        int[] nums2 = {2, 2};
        int[] intersection1 = intersection(nums1, nums2);
        System.out.println(Arrays.toString(intersection1));

        int[] nums3 = {4, 9, 5};
        int[] nums4 = {9, 4, 9, 8, 4};
        int[] intersection2 = intersection(nums3, nums4);
        System.out.println(Arrays.toString(intersection2));
    }
}
```

Answer 6)

```
public class MinimumRotatedSortedArray {
    public static int findMin(int[] nums) {
        int left = 0;
        int right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[right]) {
                // The minimum element is in the right half
                left = mid + 1;
            } else {
                // The minimum element is in the left half or mid itself
                right = mid;
            }
        }

        return nums[left];
    }

    public static void main(String[] args) {
```

```
        int[] nums1 = {3, 4, 5, 1, 2};
        int min1 = findMin(nums1);
        System.out.println("Minimum element: " + min1);

        int[] nums2 = {4, 5, 6, 7, 0, 1, 2};
        int min2 = findMin(nums2);
        System.out.println("Minimum element: " + min2);
    }
}
```

Answer 7)

```
public class FindRange {
    public static int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};

        // Search for the leftmost position of the target
        int left = searchLeft(nums, target);
        if (left == -1) {
            return result; // Target not found
        }

        // Search for the rightmost position of the target
        int right = searchRight(nums, target);

        result[0] = left;
        result[1] = right;
        return result;
    }

    private static int searchLeft(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        int index = -1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] >= target) {
                right = mid - 1;
                if (nums[mid] == target) {
                    index = mid;
                }
```

```
            } else {
                left = mid + 1;
            }
        }

        return index;
    }

    private static int searchRight(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        int index = -1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] <= target) {
                left = mid + 1;
                if (nums[mid] == target) {
                    index = mid;
                }
            } else {
                right = mid - 1;
            }
        }

        return index;
    }

    public static void main(String[] args) {
        int[] nums = {5, 7, 7, 8, 8, 10};
        int target = 8;
        int[] range = searchRange(nums, target);
        System.out.println("Range: [" + range[0] + ", " + range[1] + "]");

        target = 6;
        range = searchRange(nums, target);
        System.out.println("Range: [" + range[0] + ", " + range[1] + "]");
    }
}
```

Answer 8)

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class FindIntersection {
    public static int[] intersect(int[] nums1, int[] nums2) {
        Arrays.sort(nums1); // Sort the first array
        Arrays.sort(nums2); // Sort the second array

        List<Integer> intersection = new ArrayList<>();

        int i = 0; // Pointer for the first array
        int j = 0; // Pointer for the second array

        while (i < nums1.length && j < nums2.length) {
            if (nums1[i] < nums2[j]) {
                i++; // Move pointer for the first array
            } else if (nums1[i] > nums2[j]) {
                j++; // Move pointer for the second array
            } else {
                intersection.add(nums1[i]); // Add the common element to the intersection list
                i++; // Move both pointers to check for more common elements
                j++;
            }
        }

        // Convert the list to an array
        int[] result = new int[intersection.size()];
        for (int k = 0; k < intersection.size(); k++) {
            result[k] = intersection.get(k);
        }

        return result;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 2, 2, 1};
        int[] nums2 = {2, 2};
        int[] intersection = intersect(nums1, nums2);
        System.out.println("Intersection: " + Arrays.toString(intersection));

        int[] nums3 = {4, 9, 5};
        int[] nums4 = {9, 4, 9, 8, 4};
        intersection = intersect(nums3, nums4);
```

```
        System.out.println("Intersection: " + Arrays.toString(intersection));
    }
}
```