Longest positive number subsequence.

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

public class Main {
    public static void main(String[] args) {
        int[] nums = {1, 2, -3, 4, 5, -6, 7, 8, 9, 10};
        List<Integer> result = longestPositiveSubsequence(nums);
        System.out.println(result);
    }

    public static List<Integer> longestPositiveSubsequence(int[] nums) {
        List<Integer> result = new ArrayList<>();
        List<Integer> currentSubsequence = new ArrayList<>();

        for (int num : nums) {
            if (num > 0) {
                currentSubsequence.add(num);
            } else {
                if (currentSubsequence.size() > result.size()) {
                    result = new ArrayList<>(currentSubsequence);
                }
                currentSubsequence.clear();
            }
        }

        if (currentSubsequence.size() > result.size()) {
            result = new ArrayList<>(currentSubsequence);
        }

        return result;
    }
}
```

*Right view of a binary tree.

```
import java.util.ArrayList;
import java.util.LinkedList;
```

```
import java.util.List;
import java.util.Queue;

class TreeNode {
    int val;
    TreeNode left, right;

    public TreeNode(int val) {
        this.val = val;
        this.left = this.right = null;
    }
}

public class Main {
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(4);

        List<Integer> result = rightView(root);
        System.out.println(result);
    }

    public static List<Integer> rightView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (i == size - 1) {
                    result.add(node.val);
                }
                if (node.left != null) queue.add(node.left);
```

```
            if (node.right != null) queue.add(node.right);
        }
    }

    return result;
  }
}
```

*Move all 0s in the array to the right.

```
import java.util.Arrays;

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

  public static void moveZerosToRight(int[] nums) {
    int n = nums.length;
    int nonZeroIndex = 0;

    for (int i = 0; i < n; i++) {
      if (nums[i] != 0) {
        int temp = nums[i];
        nums[i] = nums[nonZeroIndex];
        nums[nonZeroIndex] = temp;
        nonZeroIndex++;
      }
    }
  }
}
```

*Reverse a linked list.

```
class ListNode {
  int val;
  ListNode next;
```

```
    public ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class Main {
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(4);

        ListNode reversed = reverseLinkedList(head);
        printList(reversed);
    }

    public static ListNode reverseLinkedList(ListNode head) {
        ListNode prev = null;
        ListNode current = head;

        while (current != null) {
            ListNode nextNode = current.next;
            current.next = prev;
            prev = current;
            current = nextNode;
        }

        return prev;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
        System.out.println();
    }
}
```

find a element in pivoted sorted array

```
public class Main {
    public static void main(String[] args) {
        int[] nums = {4, 5, 6, 7, 8, 9, 1, 2, 3};
        int target = 6;
        int result = searchInRotatedArray(nums, target);
        System.out.println(result);
    }

    public static int searchInRotatedArray(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

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

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

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

        return -1;
```

```
    }
}
```