Practice questions

1. 0-1 knapsack problem

```java
import java.util.*;
import java.util.Scanner;
public class Knapsack {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of items: ");
        int N = scanner.nextInt();
        System.out.print("Enter maximum weight capacity: ");
        int maxWeight = scanner.nextInt();
        int[] weights = new int[N];
        int[] values = new int[N];
        System.out.println("Enter weights of items:");
        for (int i = 0; i < N; i++) {
            weights[i] = scanner.nextInt();
        }
        System.out.println("Enter values of items:");
        for (int i = 0; i < N; i++) {
            values[i] = scanner.nextInt();
        }
        long[][] d = new long[N + 1][maxWeight + 1];
        for (int i = 0; i < N; i++) {
            for (int w = 0; w <= maxWeight; w++) {
                if (weights[i] <= w) {
                    // Exclude or include the item
                    d[i + 1][w] = Math.max(d[i][w], d[i][w - weights[i]] + values[i]);
                } else {
                    // Exclude the item
                    d[i + 1][w] = d[i][w];
                }
            }
        }
        System.out.println("Maximum value: " + d[N][maxWeight]);
    }
}
```

```
C:\Users\Personal\Downloads>javac Knapsack.java

C:\Users\Personal\Downloads>java Knapsack
Enter number of items: 3
Enter maximum weight capacity: 50
Enter weights of items:
10 20 30
Enter values of items:
60 100 120
Maximum value: 220
```

Time Complexity: O(N×W)

Space Complexity: O(N×W)

2. Floor in sorted array

```java
import java.util.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class floor {
    public static int searchInsert(List<Integer> nums, int k) {
        int s = 0;
        int e = nums.size() - 1;
        while (s <= e) {
            int mid = s + (e - s) / 2;
            if (nums.get(mid) == k) {
                return mid;
            }
            if (nums.get(mid) > k) {
                e = mid - 1;
            } else {
                s = mid + 1;
            }
        }
        return s;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the sorted list: ");
        int n = scanner.nextInt();
        List<Integer> nums = new ArrayList<>();
        System.out.println("Enter the sorted list of numbers:");
        for (int i = 0; i < n; i++) {
            nums.add(scanner.nextInt());
        }
        System.out.print("Enter the target number: ");
        int k = scanner.nextInt();
        int position = searchInsert(nums, k);
        System.out.println("The target number should be inserted at index: " + position);
        scanner.close();
    }
}
```

```
C:\Users\Personal\Downloads>javac floor.java

C:\Users\Personal\Downloads>java floor
Enter the number of elements in the sorted list: 5
Enter the sorted list of numbers:
1 3 5 6 8
Enter the target number: 7
The target number should be inserted at index: 4
```

Time Complexity: O(logN)

Space Complexity: O(N)

3. Check equal arrays

```java
import java.util.*;
import java.io.*;
import java.util.*;
class equalarray {
    public static boolean areEqual(int arr1[], int arr2[])
    {
        int N = arr1.length;
        int M = arr2.length;
        if (N != M)
            return false;
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        for (int i = 0; i < N; i++)
            if (arr1[i] != arr2[i])
                return false;
        return true;
    }
    public static void main(String[] args)
    {
        int arr1[] = { 3, 5, 2, 5, 2 };
        int arr2[] = { 2, 3, 5, 5, 2 };
        if (areEqual(arr1, arr2))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

```
C:\Users\Personal\Downloads>javac equalarray.java

C:\Users\Personal\Downloads>java equalarray
Yes
```

Time Complexity: O(N*log(N))

Space Complexity: O(1)

4. Palindrome linked list

```
import java.util.*;
class Node {
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}
class Palindromelinkedlist {
    static Node reverseList(Node head) {
        Node prev = null;
        Node curr = head;
        Node next;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
    static boolean isIdentical(Node n1, Node n2) {
        while (n1 != null && n2 != null) {
            if (n1.data != n2.data)
                return false;
            n1 = n1.next;
            n2 = n2.next;
        }
        return true;
    }
    static boolean isPalindrome(Node head) {
        if (head == null || head.next == null)
            return true;
        Node slow = head, fast = head;
        while (fast.next != null
               && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        Node head2 = reverseList(slow.next);
        slow.next = null;
        boolean ret = isIdentical(head, head2);
        head2 = reverseList(head2);
        slow.next = head2;
        return ret;
    }
    public static void main(String[] args) {
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = new Node(2);
        head.next.next.next.next = new Node(1);
        boolean result = isPalindrome(head);
        if (result)
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

```
C:\Users\Personal\Downloads>javac Palindromelinkedlist.java

C:\Users\Personal\Downloads>java Palindromelinkedlist
true
```

Time Complexity: O(n)

Space Complexity: O(1)

5. Balanced tree check

```
import java.util.*;
class Node {
    int data;
    Node left, right;
    Node(int d) {
        data = d;
        left = right = null;
    }
}
class Balancedtreecheck {
    Node root;
    boolean isBalanced(Node node) {
        int lh, rh;
        if (node == null) return true;
        lh = height(node.left);
        rh = height(node.right);
        return Math.abs(lh - rh) <= 1 && isBalanced(node.left) && isBalanced(node.right);
    }
    int height(Node node) {
        if (node == null) return 0;
        return 1 + Math.max(height(node.left), height(node.right));
    }
    public static void main(String args[]) {
        Balancedtreecheck tree = new Balancedtreecheck();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.left.left.left = new Node(8);
        if (tree.isBalanced(tree.root))
            System.out.println("Tree is balanced");
        else
            System.out.println("Tree is not balanced");
    }
}
```

```
C:\Users\Personal\Downloads>javac Balancedtreecheck.java

C:\Users\Personal\Downloads>java Balancedtreecheck
Tree is not balanced
```

Time Complexity: O(n^2)

Space Complexity: O(n)

6. Triplet sum in array

```java
import java.util.*;
import java.util.Arrays;
public class Tripletsuminarray {
    static boolean find3Numbers(int[] arr, int sum) {
        int n = arr.length;
        for (int i = 0; i < n - 2; i++) {
            for (int j = i + 1; j < n - 1; j++) {
                for (int k = j + 1; k < n; k++) {
                    if (arr[i] + arr[j] + arr[k] == sum) {
                        System.out.println("Triplet is " + arr[i] + ", " + arr[j] + ", " + arr[k]);
                        return true;
                    }
                }
            }
        }
        return false;
    }
    public static void main(String[] args) {
        int[] arr = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;
        find3Numbers(arr, sum);
    }
}
```

```
C:\Users\Personal\Downloads>javac Tripletsuminarray.java

C:\Users\Personal\Downloads>java Tripletsuminarray
Triplet is 4, 10, 8
```

Time Complexity: O(n^3)

Space Complexity: O(1)