```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class Main {
    public static void main(String[] args) {
        int[] arr = {5, 11, 15, 17, 24, 30, 36, 37, 40, 45};
        int[] arr1 = {6, 12, 19, 27, 28, 29, 33, 36, 46, 48};
        int x = 36;

        System.out.println("Cac so nguyen to trong mang la:");
        primeInArr(arr);

        System.out.println("Mang sau khi sap xep la:");
        Sort.quickSort(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));

        System.out.println("So lan xuat hien cua " + x + " trong
mang la:");
        appearIn2Arr(arr, arr1, x);

        int[] a = {1, 5, 2, 6, 3, 7, 4};
        int[] b = {4, 7, 2, 5, 8, 6, 3, 1};
        System.out.println("Mang chung dai nhat cua 2 mang
la:");
        getLongestCommonSubarray(a, b);

        System.out.println("Tong lon nhat cua mang con la:");
        sumMaxSubArray(arr);

        System.out.println("Hanoi Tower");
        HanoiTower(3, 'A', 'C', 'B');

        System.out.println("Knapsack DP");
        int[] val = {60, 100, 120};
        int[] wt = {10, 20, 30};
        int W = 50;
        System.out.println(knapsackDP(val, wt, W, val.length));

        int[][] graph = new int[][]{
                {0, 4, 0, 0, 0, 0, 0, 8, 0},
                {4, 0, 8, 0, 0, 0, 0, 11, 0},
                {0, 8, 0, 7, 0, 4, 0, 0, 2},
                {0, 0, 7, 0, 9, 14, 0, 0, 0},
                {0, 0, 0, 9, 0, 10, 0, 0, 0},
```

```
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}
};


System.out.println("Dijkstra");
dijkstra(graph, 0);

System.out.println("Bellman Ford");
bellmanFord(graph, 0);

int [][]edge={
        {0,4,0,0,5,0,0,0},
        {4,0,5,2,0,0,8,0},
        {0,5,0,0,0,0,0,0},
        {0,2,0,0,0,3,0,7},
        {5,0,0,0,0,0,2,3},
        {0,0,0,3,0,0,7,0},
        {0,8,0,0,2,7,0,0},
        {0,0,0,7,3,0,0,0}

};

int [][]edge2={
        {0,4,0,0,5,0,0,0},
        {4,0,0,8,0,0,8,0},
        {0,0,0,0,0,7,0,0},
        {0,8,0,0,0,2,0,7},
        {5,0,0,0,0,0,1,0},
        {0,0,7,2,0,0,0,4},
        {0,0,0,0,1,0,0,0},
        {0,0,0,7,0,4,0,0}

};

System.out.println("Prim MST");
primMST(edge);
System.out.println("Prim MST");
primMST(edge2);
System.out.println("Kruskal MST");
kruskalsMST(edge);
```

```
    }


    public static boolean isPrime(int n) {
        if (n <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static void primeInArr(int[] arr) {
        for (int j : arr) {
            if (isPrime(j)) {
                System.out.print(j + "\t");
            }
        }
        System.out.println();
    }

    public static void appearIn2Arr(int[] arr1, int[] arr2, int
x) {
        int[] maxArr = (arr1.length > arr2.length) ? arr1 :
arr2;
        int[] minArr = (arr1.length < arr2.length) ? arr1 :
arr2;

        int count = 0;

        for (int i = 0; i < minArr.length; i++) {
            if (minArr[i] == x) {
                count++;
            }
            if (maxArr[i] == x) {
                count++;
            }
        }

        for (int i = minArr.length; i < maxArr.length; i++) {
            if (maxArr[i] == x) {
                count++;
            }
```

```
        }

        if (count > 0) {
            System.out.println(x + " xuat hien " + count + "
lan");
        } else {
            System.out.println(x + " khong xuat hien trong
mang");
        }

    }

    public static void getLongestCommonSubarray(int[] a, int[]
b) {

        int aLength = a.length;
        int bLength = b.length;

        int[][] dp = new int[aLength + 1][bLength + 1];
        int maxLength = 0;
        int x_endIndex = 0;

        for (int i = 0; i <= aLength; i++) {

            for (int j = 0; j <= bLength; j++) {

                if (i == 0 || j == 0) {
                    dp[i][j] = 0;
                } else if (a[i - 1] == b[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;

                    if (maxLength < dp[i][j]) {

                        maxLength = dp[i][j];
                        x_endIndex = i - 1;
                    }

                } else {
                    dp[i][j] = 0;
                }
            }
        }

        int[] result = new int[maxLength];

        if (x_endIndex + 1 - (x_endIndex - maxLength + 1) >= 0)
```

```
            System.arraycopy(a, x_endIndex - maxLength + 1,
result, 0, x_endIndex + 1 - (x_endIndex - maxLength + 1));

        if (result.length > 0) {
            System.out.println(Arrays.toString(result));
        } else {
            System.out.println("Khong co day con chung dai
nhat");
        }
    }

    public static void primeFactor(int n) {
        for (int i = 2; i <= n; i++) {
            while (n % i == 0) {
                System.out.print(i + "\t");
                n /= i;
            }
        }
        System.out.println();
    }

    public static void sumMaxSubArray(int[] arr) {
        int max = Integer.MIN_VALUE;
        int sum = 0;

        for (int j : arr) {
            sum += j;
            max = Math.max(max, sum);

            if (sum < 0) {
                sum = 0;
            }
        }
        System.out.println(max);
    }

    public static int knapsackDP(int[] val, int[] wt, int W, int
n) {
        int[][] dp = new int[n + 1][W + 1];

        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= W; j++) {
                if (i == 0 || j == 0) {
                    dp[i][j] = 0;
                } else if (wt[i - 1] <= j) {
                    dp[i][j] = Math.max(val[i - 1] + dp[i - 1][j
```

```
- wt[i - 1]], dp[i - 1][j]);
                } else {
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }

        return dp[n][W];
    }

    public static void HanoiTower(int n, char from, char to,
char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from rod " + from +
" to rod " + to);
            return;
        }
        HanoiTower(n - 1, from, aux, to);
        System.out.println("Move disk " + n + " from rod " +
from + " to rod " + to);
        HanoiTower(n - 1, aux, to, from);
    }


    public static int minDistance(int[] dist, boolean[] sptSet)
{
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < dist.length; v++) {
            if (!sptSet[v] && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }
        }
        return min_index;
    }

    public static void dijkstra(int[][] graph, int src) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];

        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }
```

```
        dist[src] = 0;

        for (int i = 0; i < n - 1; i++) {
            int u = minDistance(dist, visited);
            visited[u] = true;

            for (int v = 0; v < n; v++) {
                if (!visited[v] && graph[u][v] != 0 &&
dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v])
{
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }

        System.out.println("Vertex \t\t Distance from Source");
        for (int i = 0; i < n; i++)
            System.out.println(src + "->" + i + " \t\t " +
dist[i]);
    }


    public static void bellmanFord(int[][] graph, int src) {
        int n = graph.length;
        int[] dist = new int[n];

        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
        }

        dist[src] = 0;

        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    if (graph[j][k] != 0 && dist[j] !=
Integer.MAX_VALUE && dist[j] + graph[j][k] < dist[k]) {
                        dist[k] = dist[j] + graph[j][k];
                    }
                }
            }
        }

        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
```

```
                if (graph[j][k] != 0 && dist[j] !=
Integer.MAX_VALUE && dist[j] + graph[j][k] < dist[k]) {
                    System.out.println("Graph contains negative
weight cycle");
                    return;
                }
            }
        }

        System.out.println("Vertex \t\t Distance from Source");
        for (int i = 0; i < n; i++)
            System.out.println(src + "->" + i + " \t\t " +
dist[i]);
    }

    public static void fill(int [][]graph){
        for (int i = 0; i < graph.length; i++) {
            for (int j = 0; j < graph[0].length; j++) {
                if (graph[i][j] == 0 && i != j) {
                    graph[i][j] = Integer.MAX_VALUE;
                }
            }
        }
    }
    public static int minKey(int[] key, boolean[] mstSet) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;

        for (int i = 0; i < key.length; i++) {
            if (!mstSet[i] && key[i] < min) {
                min = key[i];
                minIndex = i;
            }
        }
        return minIndex;
    }
    public static void primMST(int[][] graph) {
        fill(graph);

        int n = graph.length;
        int[] parent = new int[n];
        int[] key = new int[n];
        boolean[] mstSet = new boolean[n];

        for (int i = 0; i < n; i++) {
            key[i] = Integer.MAX_VALUE;
```

```
            mstSet[i] = false;
        }

        key[0] = 0;
        parent[0] = -1;

        for (int i = 0; i < n - 1; i++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;

            for (int v = 0; v < n; v++) {
                if (graph[u][v] != 0 && !mstSet[v] &&
graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }

        for (int i = 1; i < n; i++) {
            System.out.println(parent[i] + " - " + i + " : " +
graph[i][parent[i]]);
        }

        int sum = 0;
        for (int i = 1; i < n; i++) {
            sum += graph[i][parent[i]];
        }
        System.out.println("Total cost of MST: " + sum);
    }

    public static int find(int[] parent, int i){
        if (parent[i] == -1) {
            return i;
        }
        return find(parent, parent[i]);
    }

    public static void union(int[] parent, int x, int y){
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }

    public static void kruskalsMST(int[][] graph){
        fill(graph);
```

```
        int n = graph.length;
        int[] parent = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = -1;
        }

        int e = 0;
        int i = 0;
        while (e < n - 1) {
            int min = Integer.MAX_VALUE;
            int a = -1, b = -1;
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    if (find(parent, j) != find(parent, k) &&
graph[j][k] < min) {
                        min = graph[j][k];
                        a = j;
                        b = k;
                    }
                }
            }
            union(parent, a, b);
            System.out.println("Edge " + ++e + ": (" + a + ", "
+ b + ") cost: " + min);
        }
    }

}

class Search{
    public static void depthFirstSearch(int[][] graph, int src)
{
        int n = graph.length;
        boolean[] visited = new boolean[n];

        for (int i = 0; i < n; i++) {
            visited[i] = false;
        }

        Stack<Integer> stack = new Stack<>();
        stack.push(src);

        while (!stack.isEmpty()) {
            int u = stack.pop();
            if (!visited[u]) {
                System.out.print(u + " ");
```

```
                visited[u] = true;
            }

            for (int v = 0; v < n; v++) {
                if (graph[u][v] != 0 && !visited[v]) {
                    stack.push(v);
                }
            }
        }
    }

    public static void breadthFirstSearch(int[][] graph, int
src) {
        int n = graph.length;
        boolean[] visited = new boolean[n];

        for (int i = 0; i < n; i++) {
            visited[i] = false;
        }

        Queue<Integer> queue = new LinkedList<>();
        queue.add(src);

        while (!queue.isEmpty()) {
            int u = queue.poll();
            if (!visited[u]) {
                System.out.print(u + " ");
                visited[u] = true;
            }

            for (int v = 0; v < n; v++) {
                if (graph[u][v] != 0 && !visited[v]) {
                    queue.add(v);
                }
            }
        }
    }
}

class Sort{
    public static void bubbleSort(int[] arr){
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
```

```
                arr[j + 1] = temp;
            }
        }
    }
}

public static void selectionSort(int[] arr){
    for (int i = 0; i < arr.length; i++) {
        int min = i;
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}

public static void insertionSort(int[] arr){
    for (int i = 1; i < arr.length; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

public static void merge(int[] arr, int l, int m, int r){
    int n1 = m - l + 1;
    int n2 = r - m;

    int[] L = new int[n1];
    int[] R = new int[n2];

    for (int i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (int i = 0; i < n2; i++) {
        R[i] = arr[m + 1 + i];
    }
```

```
    int i = 0, j = 0;
    int k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

public static void mergeSort(int[] arr, int l, int r){
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

public static int partition(int[] arr, int low, int high){
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
```

```
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void quickSort(int[] arr, int low, int high){
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static void shellSort(int[] arr){
        int n = arr.length;
        for (int gap = n / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < n; i++) {
                int temp = arr[i];
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -
= gap) {
                    arr[j] = arr[j - gap];
                }
                arr[j] = temp;
            }
        }
    }


}
```