```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
public class BFS {
    public ArrayList<Integer> BFS(int v, ArrayList<ArrayList<Integer>> adj) {
        boolean[] visited = new boolean[v];
        ArrayList<Integer> ans = new ArrayList<>();
        for (int i = 0; i < v; i++) {
            if (!visited[i]) {
                bfs(i, visited, adj, ans);
            }
        }
        return ans;
    }

    public void bfs(int node, boolean[] visited, ArrayList<ArrayList<Integer>> adj,
ArrayList<Integer> ans) {
        Queue<Integer> q = new LinkedList<>();
        q.offer(node);
        visited[node] = true;
        while (!q.isEmpty()) {
            int u = q.poll();
            ans.add(u);
            for (int v : adj.get(u)) {
                if (!visited[v]) {
                    visited[v] = true;
                    q.offer(v);
                }
            }
        }

    }
    public static void main(String[] args) {
        // Sample test case
        Graph g = new Graph(5);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(2, 4);
        g.addEdge(3, 4);
        BFS bfs = new BFS();
        ArrayList<Integer> sol = bfs.BFS(g.getVertices(), g.getAdj());
        System.out.println(sol);

    }
}
```

```java
import java.util.ArrayList;
import java.util.Scanner;

public class DFS {
    public ArrayList<Integer> DFS(int v, ArrayList<ArrayList<Integer>> adj) {
        boolean visited[] = new boolean[v];
        ArrayList<Integer> res = new ArrayList<>();
        for (int i = 0; i < v; i++) {
            if (!visited[i]) {
                dfs(i, visited, adj, res);
            }
        }
        return res;
    }

    public void dfs(int node, boolean[] visited, ArrayList<ArrayList<Integer>> adj,
ArrayList<Integer> res) {
        visited[node] = true;
        res.add(node);
        for (int i : adj.get(node)) {
            if (!visited[i]) {
                dfs(i, visited, adj, res);
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of vertices: ");
        int v = scanner.nextInt();

        ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < v; i++) {
            adj.add(new ArrayList<>());
        }

        System.out.print("Enter number of edges: ");
        int e = scanner.nextInt();

        System.out.println("Enter edges (start end): ");
        for (int i = 0; i < e; i++) {
            int u = scanner.nextInt();
            int v1 = scanner.nextInt();
            adj.get(u).add(v1);
            adj.get(v1).add(u); // If the graph is undirected
        }

        DFS dfsTraversal = new DFS();
        ArrayList<Integer> result = dfsTraversal.DFS(v, adj);

        System.out.println("DFS Traversal: " + result);

        scanner.close();
    }
}
```

```java
import java.util.ArrayList;
public class Graph {
    ArrayList<ArrayList<Integer>> adj;
    int vertices;
    public Graph(int vertices){
        this.vertices = vertices;
        adj = new ArrayList<>() ;
        for(int i = 0; i < vertices; i++){
            adj.add(new ArrayList<>());
        }
    }
    public int getVertices(){
        return vertices;
    }
    public void addEdge(int v1, int v2,boolean isDirected){
        adj.get(v1).add(v2);
        if(!isDirected){
            adj.get(v2).add(v1);
        }
    }
    public void addEdge(int v1, int v2){
        addEdge(v1,v2,false);
    }
    public ArrayList<Integer> getAdj(int v){
        return adj.get(v);
    }
    public ArrayList<ArrayList<Integer>> getAdj(){
        return adj;
    }
}
```

```java
import java.util.Arrays;

public class MergeSort {
    static void mergeSort(int[] arr, int l, int r) {
        if (l >= r) return;
        int mid = l + (r - l) / 2;

        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }

    static void merge(int[] arr, int l, int mid, int r) {
        int n1 = mid - l + 1, n2 = r - mid;
        int[] left = new int[n1], right = new int[n2];

        for (int i = 0; i < n1; i++) left[i] = arr[l + i];
        for (int i = 0; i < n2; i++) right[i] = arr[mid + 1 + i];

        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) arr[k++] = (left[i] < right[j]) ? left[i++] : right[j++];
        while (i < n1) arr[k++] = left[i++];
        while (j < n2) arr[k++] = right[j++];
    }

    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2};
        mergeSort(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }
}
```

```java
import java.util.Arrays;
public class QuickSort {
    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivot = partition(arr, low, high);
            quickSort(arr, low, pivot - 1);
            quickSort(arr, pivot + 1, high);
        }
    }
    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) swap(arr, ++i, j);
        }
        swap(arr, i + 1, high);
        return i + 1;
    }
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2};
        quickSort(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }
}
```