Breadth-First Search (BFS) Algorithm:

Choose a starting vertex, mark it as visited, and add it to a queue.
While the queue is not empty:
Dequeue a vertex from the queue and process it (e.g., print its value).
Enqueue all its adjacent vertices that have not been visited and mark them as visited.
Repeat step 2 until the queue is empty.

Here's some pseudocode to illustrate the algorithm:

```
// Here, Graph is the graph that we already have and X is the source node
Breadth_First_Search( Graph, X ):
   Let Q be the queue
   Q.enqueue( X )     // Inserting source node X into the queue
   Mark X node as visited.

   While ( Q is not empty )
      Y = Q.dequeue( )     // Removing the front node from the queue

   Process all the neighbors of Y, For all the neighbors Z of Y
   If Z is not visited:
      Q. enqueue( Z )     // Stores Z in Q
      Mark Z as visited
```

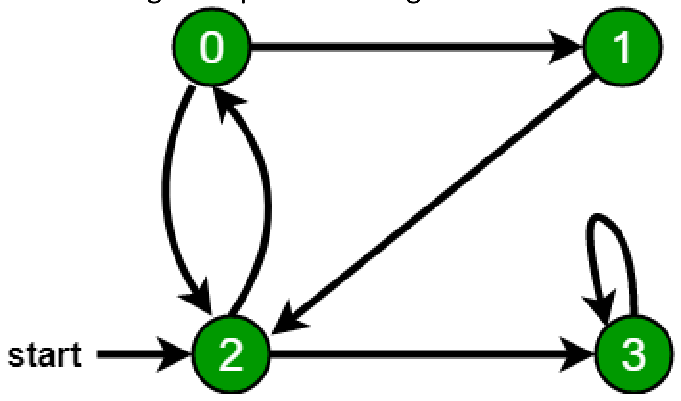
In this algorithm, we use a queue to keep track of the vertices to be visited. We start by choosing a starting vertex, marking it as visited, and adding it to the queue. Then, while the queue is not empty, we dequeue a vertex from the queue, process it (e.g., print its value), and enqueue all its adjacent vertices that have not been visited yet, marking them as visited.

The BFS algorithm is commonly used to traverse and search graphs, and it guarantees to visit all the vertices in the graph, starting from a given vertex, in a breadth-first order. It can also be used to find the shortest path between two vertices in an unweighted graph.

Exercise 1 : Complete the following code and calculate its complexity.

```
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency
// list representation
class Graph {

    // No. of vertices
private int V;

// Adjacency Lists
private LinkedList<Integer> adj[];

// Constructor
Graph(int v)
{
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
        adj[i] = new LinkedList();
}

// Function to add an edge into the graph
void addEdge(int v, int w) { adj[v].add(w); }

// prints BFS traversal from a given source s
void BFS(int s)
{
    // Mark all the vertices as not visited(By default
    // set as false)
    boolean visited[] = new boolean[V];

    // Create a queue for BFS
    LinkedList<Integer> queue
        = new LinkedList<Integer>();

    // Mark the current node as visited and enqueue it
    ......
   }
}
```

Exercise 2 : Consider the following example for testing:



Exercise 3: What is about the disconnected graphs ?

Exercise 4:
Implement a variant version of the BFS for shortest path search.

Indication : The idea is to use a modified version of BFS in which we keep storing the predecessor of a given vertex while doing the breadth-first search.
We first initialize an array dist[0, 1, …., v-1] such that dist[i] stores the distance of vertex i from the source vertex and array pred[0, 1, ….., v-1] such that pred[i] represents the immediate predecessor of the vertex i in the breadth-first search starting from the source.