Approach to solve BFS :

- Given : Graph (Adjacency List), number of nodes
- Declare :
    - Empty queue
    - Visited vector having size of number of nodes [1-n : 1 based nodes, 0-n-1 : 0 based nodes]
    - Result vector
- Insert the first node into the queue and mark it as visited in the visited vector
- Loop until the queue becomes empty :
    - Extract the first element of the queue and insert it into the resultant vector
    - Pop the first element from the queue
    - Traverse for all the adjacent elements of the given node using the adjacency list :
        - If not visited then :
            - Insert the adjacent node into the queue
            - Mark the node as visited
- Return the resultant vector

Function Code :

```
vector<int> BFS(int n, vector<int> adj[]) {
        // Declaring :
        // Visited vector containing all 0 elements of size n
        vector<int> visited(n,0);
        // Empty queue
        queue<int> q;
        // Result vector
        vector<int> result;

        // Inserting first element into the queue and marking it as visited
        q.push(0);
        visited[0] = 1;

        // Checking if q is not empty and traversing through levels
        while(!q.empty())
        {
            // Extracting the first element of the queue
            int node = q.front();
            // Inserting the first element into the result vector
            result.push_back(node);
            // Popping the element from the queue
            q.pop();
```

```
            // traversing through the adjacent elements of the vector

            for(int adjacentNode:adj[node])
            {
                if(!visited[adjacentNode])
                {
                    visited[adjacentNode] = 1;
                    q.push(adjacentNode);
                }
            }

        }
        //returning the result vector
        return result;
```

Time Complexity

O(V+E)