**Assignment by**

**Atiq Israk Niloy**

**2016100000039**

In the code provided for quicksort, if the elements of input become same the program crashes because we weren't partitioning properly.

We were finding pivot according to the size of the array not taking a look at the array elements if there are elements smaller or greater than the pivot in left and right sides.

So, in the modified code, we have taken a function "***partition***" to make the element pivot which will be in the center of the array where left array elements will be smaller than the pivot and right elements will be greater. After that we will get a perfect array each time to run quicksort in the array.

**Modified Code:**

```
#include <stdio.h>
int array[1000],low,high,temp;
int partition (int low, int high){
   int pivot = array[high]; // pivot
   int i = (low - 1); // Index of smaller element
   for (int j = low; j <= high- 1; j++){
      // If current element is smaller than or equal to pivot
      if (array[j] <= pivot) //exit if there is no or 1 element in the segment
{
         i++; // increment index of smaller element
      //swap(&array[i], &array[j]);
         temp=array[i];
         array[i]=array[j];
         array[j]=temp;}
   }
   //swap(&array[i + 1], &array[high]);
   temp=array[i+1];
   array[i+1]=array[high];
   array[high]=temp;
   return (i + 1);
}
void quicksort(int pivot, int last) {
   if (pivot < last) {
```

```
      int part = partition(pivot, last);
      quicksort(pivot, part - 1);
      quicksort(part + 1, last);}
}
int main(){
   int n,i;
   printf("Input the size of the array :");
   scanf("%d",&n);
   printf("\n\nEnter each element: \n\n");
   for(i=0; i<n; i++)
   {scanf("%d",&array[i]);}
   quicksort(0,n-1);
   printf("\n\nAfter Sorting: \n\n");
   for(i=0; i<n; i++){printf("%d ",array[i]);}
   return 0;
}
```

## Finding shortest path by BFS Using Queue:

```
#include<bits/stdc++.h>
using namespace std;
#define Max 1009 //Let the max number of nodes be 1009
vector<int> graph[Max],path; //graph define
bool visit[Max]; //store visited nodes
int parent[Max]; //store parent nodes

void bfs(int source, int dest){
   queue<int> q; //declaring queue
   q.push(source); //pushing source in the queue
   visit[source] = true; //storing source in the visited array
   parent[source]  = -1; //source has no parent = -1

   while(q.empty() == false) //until the queue becomes empty
   {
      int u = q.front(); //front nilam
      q.pop(); //front pop kore dilam
      for(int i = 0; i < graph[u].size(); i++) //sob adjacent e gelam
      {
         int v = graph[u][i]; //adjacent detection
         if(visit[v] == false){
            parent[v] = u;
            if(v == dest)return; //v dest hole break
            q.push(v); //na hole push and visited
```

```
            visit[v] = true;
          }
        }
    }
}
void getPath(int node){
    if(node == -1)return; //source er parent -1 tai break
    path.push_back(node);
    getPath(parent[node]);  //destination er parent k call korchi
}
void printPath(int node){
    getPath(node);
    reverse(path.begin(), path.end());
    cout << "Path :";
    for(int u : path){
        cout << " " << u;
    }
    cout << endl;
}
int main(){
    int vertices, edges;
    cout << "Number of vertices: ";
    cin >> vertices;
    cout << "Number of edges: ";
    cin >> edges;
    for(int i = 1; i <= edges; i++){
        int u, v;
        cout << "Input connected nodes: ";
        cin >> u >> v;
        graph[u].push_back(v); //defining bidirectional
        graph[v].push_back(u); //defining bidirectional
    }
    int source, dest;
    cout << "Enter Source: ";
    cin >> source;
    cout << "Enter destination: ";
    cin >> dest; //taking source and destination as input

    bfs(source, dest);
    cout << "Shortest Path: ";
    printPath(dest);
    return 0;
}
```

**Source:**
QuickSort - https://github.com/niloyniil/AdvancedAlgorithm/blob/master/Quicksort%202.0.c
BFS - https://github.com/niloyniil/AdvancedAlgorithm/blob/master/BFS%20using%20queue.cpp