Sorting Algorithms

Bubble Sort:

```
void print_arr(int DATA[], int N){
    int i;
    for(i=0; i<N; i++){
        cout << DATA[i] << " " ;
    }
    cout << endl;
}

void bubble_sort(int DATA[], int N){
    int i, k;
    int temp;
    for(k=0; k<N; k++){
        int PTR = 0;
        while(PTR <= N-k){
            if(DATA[PTR] > DATA[PTR+1]){
                temp = DATA[PTR];
                DATA[PTR] = DATA [PTR+1];
                DATA [PTR+1] = temp;
            }
            PTR = PTR + 1;
        }
    }
}

int main(){
            int DATA[10] = {48, 78, 95, 5, 21, 10, 56, 12, 3, 45};
            cout << "Before sorting: " << endl;
            print_arr(DATA, 10);
            bubble_sort(DATA, 10);

            cout << "After sorting: " << endl;
            print_arr(DATA, 10);
return 0;
}
```

Quick Sort:

```
void swap(int* a, int* b){
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high){
    int pivot = arr[high];
    int _end = (low - 1);

    for (int start = low; start < high ; start++) {
        if (arr[start] < pivot) {
            _end++;
            swap(&arr[_end], &arr[start]);
        }
    }
    swap(&arr[_end + 1], &arr[high]);
    return (_end + 1);
}

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);
    }
}

void printArray(int arr[], int size){
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main(){
    int arr[] = {10, 7, 8, 9, 1, 25, 5, 52, 45, 98, 36, 45, 74, 52};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Before sorting: " << endl;
    printArray(arr, n);
    quickSort(arr, 0, n - 1);
    cout << "After sorting: \n";
    printArray(arr, n);
return 0;
}
```

Merge Sort:

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}


void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}


int main(){
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    int arr[] = {12, 11, 13, 5, 6, 7, 56, 5, 1, 0, 20, 2, 3, 5, 4, 2, 8, 9, 7 ,45, 1, 15, 3};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);


return 0;
}
```

__Insertion sort:__

```
void swap(int *xp, int *yp){
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}

void selectionSort(int arr[], int n){
        int i, j, min_idx;

        // One by one move boundary of unsorted subarray
        for (i = 0; i < n-1; i++){
                // Find the minimum element in unsorted array
                min_idx = i;
                for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main(){
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    int arr[] = {64, 25, 12, 89,  98, 65, 12, 11, 10, 2, 25, 1, 0, 3, 22, 33, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Before sorting: " << endl;
    printArray(arr, n);

    selectionSort(arr, n);

    cout << "After sorting: " << endl;
    printArray(arr, n);

return 0;
}
```

Heap sort:

```
// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
   int largest = i; // Initialize largest as root
   int l = 2*i + 1; // left = 2*i + 1
   int r = 2*i + 2; // right = 2*i + 2
```

```
    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
```

```
}



int main(){
            //freopen("in.txt", "r", stdin);
            //freopen("out.txt", "w", stdout);

    int arr[] = {12, 11, 13, 5, 6, 7, 56, 5, 1, 0, 20, 2, 3, 5, 4, 2, 8, 9, 7 ,45, 1, 15, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Before sorting: " << endl;
    printArray(arr, n);
    heapSort(arr, n);

    cout << "After sorting: " << endl;
    printArray(arr, n);



return 0;
}
```

Topological sort:

```
// Class to represent a graph
class Graph
{
    int V;    // No. of vertices'

    // Pointer to an array containing adjacency listsList
    list<int> *adj;

    // A function used by topologicalSort
    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);
public:
    Graph(int V);   // Constructor

     // function to add an edge to graph
    void addEdge(int v, int w);
    // prints a Topological Sort of the complete graph
    void topologicalSort();
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}

// A recursive function used by topologicalSort
void Graph::topologicalSortUtil(int v, bool visited[],
                                stack<int> &Stack)
{
    // Mark the current node as visited.
    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            topologicalSortUtil(*i, visited, Stack);

    // Push current vertex to stack which stores result
    Stack.push(v);
}

// The function to do Topological Sort. It uses recursive
// topologicalSortUtil()
void Graph::topologicalSort()
{
    stack<int> Stack;

    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    // Call the recursive helper function to store Topological
    // Sort starting from all vertices one by one
    for (int i = 0; i < V; i++)
      if (visited[i] == false)
        topologicalSortUtil(i, visited, Stack);

    // Print contents of stack
    while (Stack.empty() == false)
    {
        cout << Stack.top() << " ";
        Stack.pop();
    }
}


int main(){
            //freopen("in.txt", "r", stdin);
            //freopen("out.txt", "w", stdout);
                // Create a graph given in the above diagram
    Graph g(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1);

    cout << "Following is a Topological Sort of the given graph \n";
    g.topologicalSort();


return 0;
}
```

Radix sort:

```
// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)
{
    int output[n]; // output array
    int i, count[10] = {0};

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[ (arr[i]/exp)%10 ]++;

    // Change count[i] so that count[i] now contains actual
    //  position of this digit in output[]
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--)
    {
        output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];
        count[ (arr[i]/exp)%10 ]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m/exp > 0; exp *= 10)
```

```
        countSort(arr, n, exp);
}

// A utility function to print an array
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}


int main(){
            //freopen("in.txt", "r", stdin);
            //freopen("out.txt", "w", stdout);

            int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);


    cout << "Before sorting: " << endl;
    print(arr, n);
    radixsort(arr, n);

    cout << "After sorting: " << endl;
    print(arr, n);



return 0;
}
```

Shell sort:

```
/* function to sort arr using shellSort */
int shellSort(int arr[], int n)
{
    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap sorted
            // save a[i] in temp and make a hole at position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until the correct
            // location for a[i] is found
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            //  put temp (the original a[i]) in its correct location
            arr[j] = temp;
        }
    }
    return 0;
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```

Selection sort:

```
void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}



void selectionSort(int arr[], int n){
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)  {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}



void printArray(int arr[], int size){
    int i;
    for (i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}



int main(){
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Before sorting: ";
    printArray(arr, n);
    selectionSort(arr, n);
    cout << "After sorting : ";
    printArray(arr, n);
    return 0;
}
```

BFS:

```
///** -214,74,83,648 */

#include<stdio.h>
#include<iostream>
#include<queue>

using namespace std;

#define WHITE 1
#define GRAY 2
#define BLACK 3

int adj[100][100];
int color[100];
int parent[100];
int dis[100];

int node, edge;

void bfs(int startingNode)
{
    for (int i=0; i < node; ++i)
    {
        color[i] = WHITE;
        dis[i] = INT_MIN;
        parent[i] = -1;
    }

    dis[startingNode] = 0;
    parent[startingNode] = -1;

    queue<int> q;
    q.push(startingNode);

    while(!q.empty())
    {
        int x;
        x = q.front();
        q.pop();
        color[x] = GRAY;
        printf("%d ", x);

        for (int i=0; i<node; ++i)
        {
            if (adj[x][i] == 1)
            {
                if(color[i] == WHITE)
                {
                    dis[i] = dis[x] + 1;
                    parent[i] = x;
                    q.push(i);

                }
            }

        }
        color[x] = BLACK;
    }

}



int main()
{
    freopen("in.txt","r",stdin);
    scanf("%d %d", &node, &edge);

    int n1, n2;

    for (int i=0; i<edge; i++)
    {
        scanf("%d %d", &n1, &n2);
        adj[n1][n2]=1;
        adj[n2][n1]=1;
    }

    bfs(0);
    return 0;
}
/**

input:
8 7
0 1
0 2
1 3
2 4
2 5
3 6
3 7

*/
```

DFS:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>

#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <string>
#include <sstream>
#include <queue>
#include <iterator>

#include <iostream>


using namespace std;
```

```
#define WHITE 1
#define GRAY 2
#define BLACK 3

int adj[100][100];
int color[100];
int node, edge;


void dfsVisit(int x){
   color[x] = GRAY;

    printf("%d ", x);
   for(int i=0; i<node; i++){
        if(adj[x][i] == 1){
            if(color[i] == WHITE){
                dfsVisit(i);
            }
        }
   }
   color[x] = BLACK;
}


void dfs(){
    for(int i=0; i<node; i++){
        color[i] = WHITE;
    }
    for(int i=0; i<node; i++){
        if(color[i]==WHITE){
            dfsVisit(i);
        }
    }
}



int main(){
            freopen("in.txt", "r", stdin);
            //freopen("out.txt", "w", stdout);

            scanf("%d %d", &node, &edge);
            int n1, n2;
    for (int i=0; i<edge; i++)
    {
        scanf("%d %d", &n1, &n2);
        adj[n1][n2]=1;
        adj[n2][n1]=1;
    }
    dfs();


return 0;
}
```