Code for Bellman Ford Algorithm:

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <limits.h>

typedef struct {

    int src;

    int dest;

    int wt;

} Edge;

void *bellmanFord*(Edge\* graph, int V, int E, int src) {

    int\* dist = (int\*)*malloc*(V \* sizeof(int));

*for* (int i = 0; i < V; i++) {

        dist[i] = *INT\_MAX*;

    }

    dist[src] = 0;

*for* (int i = 0; i < V - 1; i++) {

*for* (int j = 0; j < E; j++) {

*if* (dist[graph[j].src] != *INT\_MAX* && dist[graph[j].src] + graph[j].wt < dist[graph[j].dest]) {

                dist[graph[j].dest] = dist[graph[j].src] + graph[j].wt;

            }

        }

    }

*printf*("Vertex Distance from Source\n");

*for* (int i = 0; i < V; i++) {

*printf*("%d \t\t %d\n", i, dist[i]);

    }

*free*(dist);

}

int *main*() {

    int V = 5;

    int E = 8;

    Edge\* graph = (Edge\*)*malloc*(E \* sizeof(Edge));

    graph[0].src = 0; graph[0].dest = 1; graph[0].wt = -1;

    graph[1].src = 0; graph[1].dest = 2; graph[1].wt = 4;

    graph[2].src = 1; graph[2].dest = 2; graph[2].wt = 3;

    graph[3].src = 1; graph[3].dest = 3; graph[3].wt = 2;

    graph[4].src = 1; graph[4].dest = 4; graph[4].wt = 2;

    graph[5].src = 3; graph[5].dest = 2; graph[5].wt = 5;

    graph[6].src = 3; graph[6].dest = 1; graph[6].wt = 1;

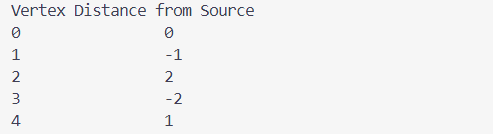
    graph[7].src = 4; graph[7].dest = 3; graph[7].wt = -3;

*bellmanFord*(graph, V, E, 0);

*return* 0;

}

Output



Code for Prim’s Algorithm:

#*include* <stdio.h>

#*include* <limits.h>

#*define* *N* 100

int *minKey*(int key*[]*, int mst*[]*, int vertices) {

    int min = *INT\_MAX*, minIndex;

*for* (int i = 0; i < vertices; i++) {

*if* (!mst[i] && key[i] < min) {

            min = key[i];

            minIndex = i;

        }

    }

*return* minIndex;

}

void *printMST*(int parent*[]*, int graph[*N*][*N*], int vertices) {

*printf*("Edge \tWeight\n");

    int cost  = 0;

*for* (int i = 1; i < vertices; i++) {

        cost += graph[i][parent[i]];

*printf*("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);

    }

*printf*("Minimum Cost of Spanning Tree : %d", cost);

}

void *primMST*(int graph[*N*][*N*], int vertices) {

    int parent[*N*];

    int key[*N*];

    int mstSet[*N*];

*for* (int i = 0; i < vertices; i++) {

        key[i] = *INT\_MAX*;

        mstSet[i] = 0;

    }

    key[0] = 0;

    parent[0] = -1;

*for* (int count = 0; count < vertices - 1; count++) {

        int u = *minKey*(key, mstSet, vertices);

        mstSet[u] = 1;

*for* (int v = 0; v < vertices; v++) {

*if* (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

                parent[v] = u;

                key[v] = graph[u][v];

            }

        }

    }

*printMST*(parent, graph, vertices);

}

int *main*() {

    int vertices;

*printf*("Input the number of vertices: ");

*scanf*("%d", &vertices);

    int graph[*N*][*N*];

*printf*("Input graph:\n");

*for* (int i = 0; i < vertices; i++) {

*for* (int j = 0; j < vertices; j++) {

*scanf*("%d", &graph[i][j]);

        }

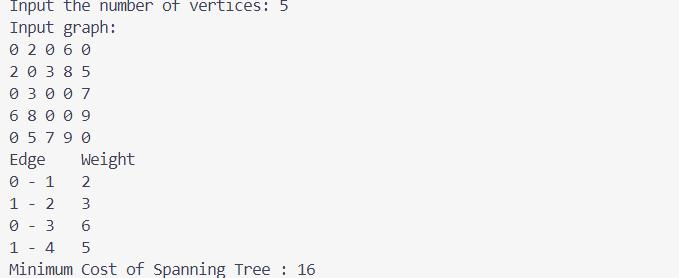
    }

*primMST*(graph, vertices);

*return* 0;

}

Output



Code for Kruskal’s Algorithm:

#*include* <stdio.h>

#*include* <stdlib.h>

typedef struct {

    int src;

    int dest;

    int wt;

} Edge;

int\* parent;

int\* rank;

void *init*(int V) {

    parent = (int\*)*malloc*(V \* sizeof(int));

    rank = (int\*)*malloc*(V \* sizeof(int));

*for* (int i = 0; i < V; i++) {

        parent[i] = i;

        rank[i] = 0;

    }

}

int *find*(int node) {

*if* (parent[node] != node) {

        parent[node] = *find*(parent[node]);

    }

*return* parent[node];

}

void *unionSets*(int u, int v) {

    int rootU = *find*(u);

    int rootV = *find*(v);

*if* (rootU != rootV) {

*if* (rank[rootU] > rank[rootV]) {

            parent[rootV] = rootU;

        } *else* *if* (rank[rootU] < rank[rootV]) {

            parent[rootU] = rootV;

        } *else* {

            parent[rootV] = rootU;

            rank[rootU]++;

        }

    }

}

int *compareEdges*(*const* void\* a, *const* void\* b) {

*return* ((Edge\*)a)->wt - ((Edge\*)b)->wt;

}

void *kruskalsMST*(Edge\* edges, int E, int V) {

*init*(V);

*qsort*(edges, E, sizeof(Edge), *compareEdges*);

    int ans = 0;

    int count = 0;

*for* (int i = 0; count < V - 1; i++) {

        Edge e = edges[i];

        int parA = *find*(e.src);

        int parB = *find*(e.dest);

*if* (parA != parB) {

*unionSets*(e.src, e.dest);

            ans += e.wt;

            count++;

        }

    }

*printf*("MST cost = %d\n", ans);

*free*(parent);

*free*(rank);

}

int *main*() {

    int V = 5;

    int E = 7;

    Edge\* edges = (Edge\*)*malloc*(E \* sizeof(Edge));

    edges[0] = (Edge){0, 1, 10};

    edges[1] = (Edge){0, 2, 6};

    edges[2] = (Edge){0, 3, 5};

    edges[3] = (Edge){1, 3, 15};

    edges[4] = (Edge){2, 3, 4};

    edges[5] = (Edge){1, 2, 5};

    edges[6] = (Edge){2, 4, 9};

*kruskalsMST*(edges, E, V);

*free*(edges);

*return* 0;

}

Output



Code for Fractional Knapsack:

#*include* <stdio.h>

void *swap*(float arr*[]*[2], int i, int j) {

    float temp0 = arr[i][0];

    float temp1 = arr[i][1];

    arr[i][0] = arr[j][0];

    arr[i][1] = arr[j][1];

    arr[j][0] = temp0;

    arr[j][1] = temp1;

}

int *partition*(float arr*[]*[2], int low, int high) {

    int i = low + 1;

    int j = high;

    int pivot = low;

*while* (i <= j) {

*while* (i <= high && arr[i][1] <= arr[pivot][1]) {

            i++;

        }

*while* (j >= low && arr[j][1] > arr[pivot][1]) {

            j--;

        }

*if* (i < j) {

*swap*(arr, i, j);

        }

    }

*swap*(arr, pivot, j);

*return* j;

}

void *quickSort*(float arr*[]*[2], int low, int high) {

*if* (low >= high) {

*return*;

    }

    int pIdx = *partition*(arr, low, high);

*quickSort*(arr, low, pIdx - 1);

*quickSort*(arr, pIdx + 1, high);

}

void *fractionalKnapsack*(int b*[]*, int wt*[]*, int W, int n) {

    float val[n][2];

*for* (int i = 0; i < n; i++) {

        val[i][0] = i;

        val[i][1] = b[i] / (float)wt[i];

    }

*quickSort*(val, 0, n - 1);

    float finalVal = 0.0;

    int w = W;

*for* (int i = n - 1; i >= 0; i--) {

        int idx = (int)val[i][0];

*if* (wt[idx] <= w) {

            w -= wt[idx];

            finalVal += b[idx];

        } *else* {

            finalVal += (b[idx] \* (float)w / wt[idx]);

*break*;

        }

    }

*printf*("Maximum value of knapsack = %.2f\n", finalVal);

}

int *main*() {

    int b*[]* = {60, 100, 120};

    int wt*[]* = {10, 20, 30};

    int W = 50;

*fractionalKnapsack*(b, wt, W, 3);

*return* 0;

}

Output



Code for Heap Sort:

#*include* <stdio.h>

void *swap*(int \*a, int \*b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void *heapify*(int arr*[]*, int n, int i) {

    int largest = i;

    int left = 2\*i+1;

    int right = 2\*i+2;

*if* (left < n && arr[left] > arr[largest])

        largest = left;

*if* (right < n && arr[right] > arr[largest])

        largest = right;

*if* (largest != i) {

*swap*(&arr[i], &arr[largest]);

*heapify*(arr, n, largest);

    }

}

void *heapSort*(int arr*[]*, int n) {

*for* (int i = n/2-1; i>=0; i--)

*heapify*(arr, n, i);

*for* (int i=n-1; i>=0; i--) {

*swap*(&arr[0], &arr[i]);

*heapify*(arr, i, 0);

    }

}

void *printArray*(int arr*[]*, int n) {

*for* (int i = 0; i < n; i++)

*printf*("%d ", arr[i]);

*printf*("\n");

}

int *main*() {

    int arr*[]* = {12, 11, 13, 5, 6, 7};

    int n = sizeof(arr) / sizeof(arr[0]);

*printf*("Original array:\n");

*printArray*(arr, n);

*heapSort*(arr, n);

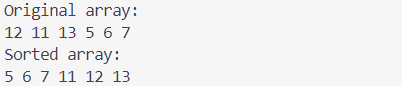
*printf*("Sorted array:\n");

*printArray*(arr, n);

*return* 0;

}

Output



Code for Non Recursive Merge Sort:

#*include* <stdio.h>

void *mergeSort*(int a*[]*, int n);

void *merge*(int arr*[]*, int si, int mid, int ei);

void *printArray*(int arr*[]*, int n);

int *main*(){

    int arr*[]* = {4, 6, 2, 5, 7, 9, 1, 3};

    int len = sizeof(arr)/sizeof(arr[0]);

*printf*("Original array:\n");

*printArray*(arr, len);

*mergeSort*(arr, len-1);

*printf*("Sorted array:\n");

*printArray*(arr, len);

*return* 0;

}

void *mergeSort*(int arr*[]*, int n){

    int p, i, s, e, mid;

*for*(p=2; p<=n; p\*=2){

*for*(i=0; i+p-1<=n; i+=p){

            s = i;

            e = i+p-1;

            mid = (e+s)/2;

*merge*(arr, s, mid, e);

        }

    }

*if*(p/2 < n){

*merge*(arr, 0, p/2-1, n-1);

    }

}

void *merge*(int arr*[]*, int si, int mid, int ei){

    int temp[ei-si+1];

    int i = si;

    int j = mid+1;

    int k = 0;

*while*(i<=mid && j<=ei){

*if*(arr[i]<arr[j]){

            temp[k++]=arr[i++];

        }

*else*{

            temp[k++] = arr[j++];

        }

    }

*while*(i<=mid){

        temp[k++] = arr[i++];

    }

*while*(j<=ei){

        temp[k++] = arr[j++];

    }

*for* (k = 0, i = si; k < ei - si + 1; k++, i++) {

    arr[i] = temp[k];

    }

}

void *printArray*(int arr*[]*, int n) {

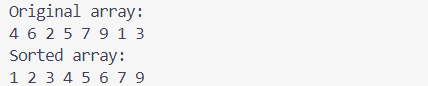
*for* (int i = 0; i < n; i++)

*printf*("%d ", arr[i]);

*printf*("\n");

}

Output



Code for Non Recursive Quick Sort:

#*include* <stdio.h>

#*define* *MAX* 100

void *swap*(int \*a, int \*b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int *partition*(int arr*[]*, int low, int high) {

    int pivot = arr[high];

    int i = low - 1;

*for* (int j = low; j < high; j++) {

*if* (arr[j] <= pivot) {

            i++;

*swap*(&arr[i], &arr[j]);

        }

    }

*swap*(&arr[i + 1], &arr[high]);

*return* (i + 1);

}

void *quickSort*(int arr*[]*, int low, int high) {

    int stack[*MAX*];

    int top = -1;

    stack[++top] = low;

    stack[++top] = high;

*while* (top >= 0) {

        high = stack[top--];

        low = stack[top--];

        int pivot = *partition*(arr, low, high);

*if* (pivot - 1 > low) {

            stack[++top] = low;

            stack[++top] = pivot - 1;

        }

*if* (pivot + 1 < high) {

            stack[++top] = pivot + 1;

            stack[++top] = high;

        }

    }

}

void *printArray*(int arr*[]*, int n) {

*for* (int i = 0; i < n; i++)

*printf*("%d ", arr[i]);

*printf*("\n");

}

int *main*() {

    int arr*[]* = {4, 6, 2, 5, 7, 9, 1, 3};

    int n = sizeof(arr) / sizeof(arr[0]);

*printf*("Original array:\n");

*printArray*(arr, n);

*quickSort*(arr, 0, n - 1);

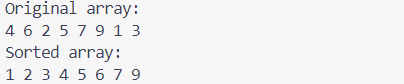
*printf*("Sorted array:\n");

*printArray*(arr, n);

*return* 0;

}

Output



Code for N Queens Problem:

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <math.h>

#*define* *MAX* 8

typedef struct {

    int \*C;

    int no\_queen;

} BOARD;

void *initialisation*(BOARD \*, int);

void *display\_board*(BOARD, int, int \*);

int *is\_safe*(BOARD, int, int);

void *n\_queen*(BOARD \*, int, int, int \*);

void *initialisation*(BOARD \*B, int n) {

    B->no\_queen = n;

    B->C = (int \*)*malloc*(sizeof(int) \* (n + 1));

*for* (int i = 1; i <= n; i++)

        B->C[i] = -1;

}

void *display\_board*(BOARD B, int n, int \*a\_sol\_no) {

*printf*("\n\n Solution %d ", ++(\*a\_sol\_no));

*for* (int i = 1; i <= n; i++) {

*printf*("\n");

*for* (int j = 1; j <= n; j++) {

*if* (B.C[i] == j)

*printf*(" Q");

*else*

*printf*(" X");

        }

    }

}

int *is\_safe*(BOARD B, int x, int y) {

*for* (int i = 1; i < x; i++) {

*if* (B.C[i] == y || *abs*(x - i) == *abs*(y - B.C[i]))

*return* 0;

    }

*return* 1;

}

void *n\_queen*(BOARD \*B, int k, int n, int \*a\_sol\_no) {

*for* (int j = 1; j <= n; j++) {

*if* (*is\_safe*(\*B, k, j)) {

            B->C[k] = j;

*if* (k == n)

*display\_board*(\*B, n, a\_sol\_no);

*else*

*n\_queen*(B, k + 1, n, a\_sol\_no);

        }

    }

}

int *main*() {

    BOARD \*p;

    int sol\_no = 0;

    p = (BOARD \*)*malloc*(sizeof(BOARD));

*initialisation*(p, 8);

*n\_queen*(p, 1, 8, &sol\_no);

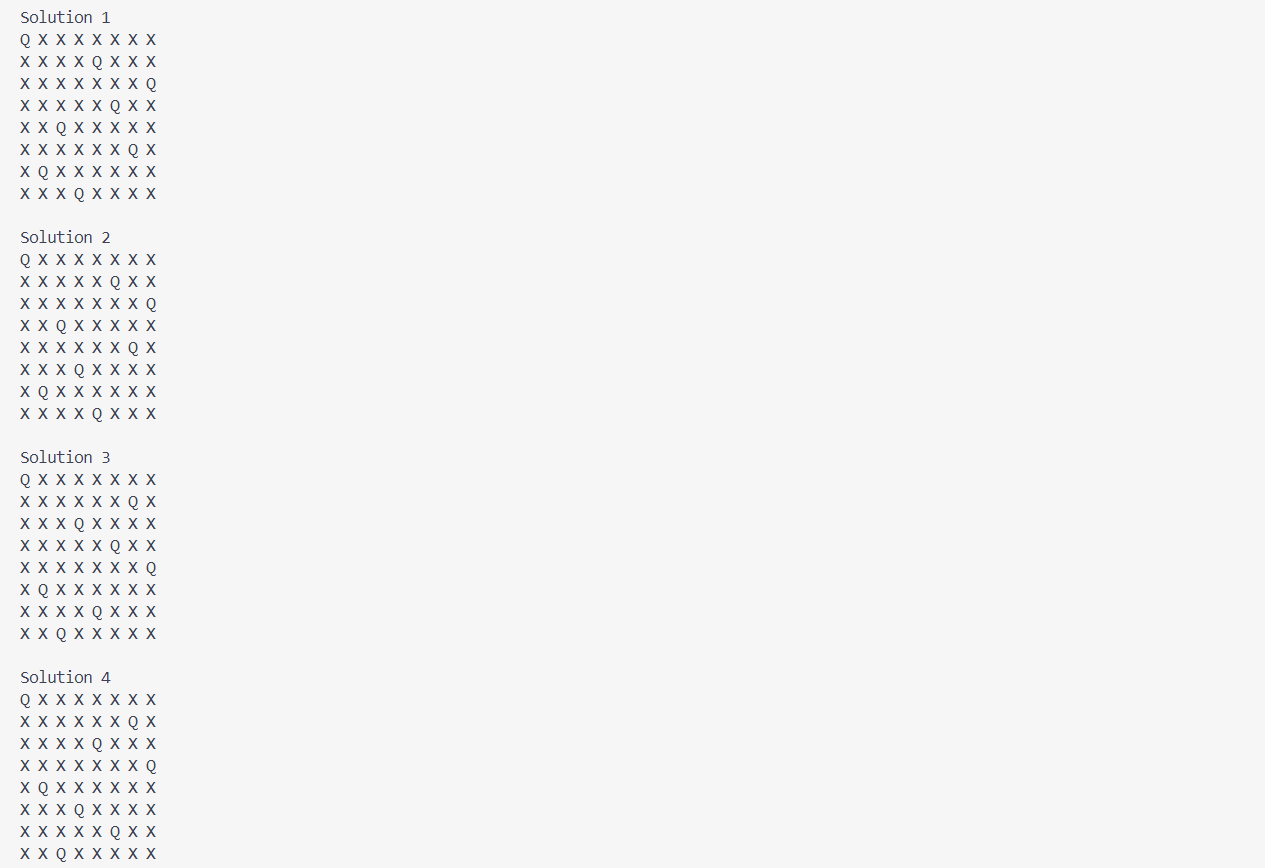
*free*(p->C);

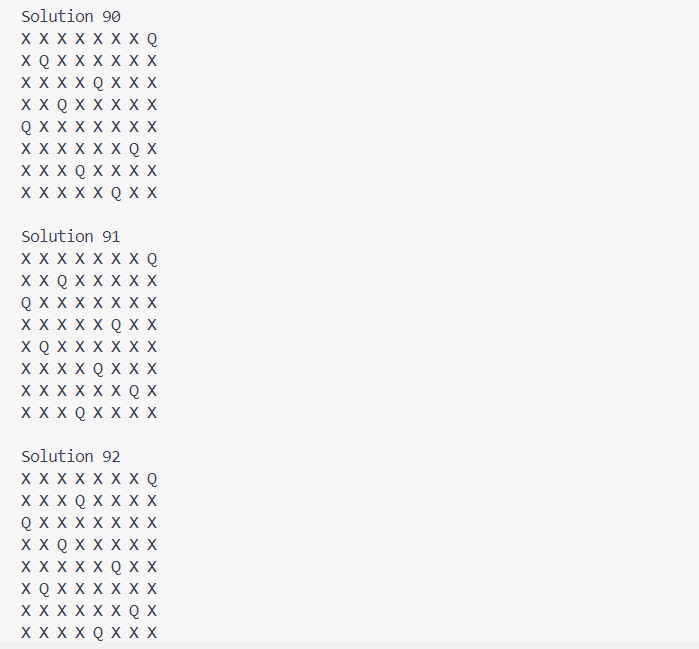
*free*(p);

*return* 0;

}

Output:





Code for Floyd algorithm:

#*include* <stdio.h>

#*define* *n* 4

#*define* *INF* 999

void *printMatrix*(int matrix*[]*[*n*]);

void *floydWarshall*(int graph*[]*[*n*]) {

    int matrix[*n*][*n*], i, j, k;

*for* (i = 0; i < *n*; i++)

*for* (j = 0; j < *n*; j++)

    matrix[i][j] = graph[i][j];

*for* (k = 0; k < *n*; k++) {

*for* (i = 0; i < *n*; i++) {

*for* (j = 0; j < *n*; j++) {

*if* (matrix[i][k] + matrix[k][j] < matrix[i][j])

                    matrix[i][j] = matrix[i][k] + matrix[k][j];

            }

        }

    }

*printMatrix*(matrix);

}

void *printMatrix*(int matrix*[]*[*n*]) {

*for* (int i = 0; i < *n*; i++) {

*for* (int j = 0; j < *n*; j++) {

*if* (matrix[i][j] == *INF*)

*printf*("%4s", "INF");

*else*

*printf*("%4d", matrix[i][j]);

        }

*printf*("\n");

    }

}

int *main*() {

    int graph[*n*][*n*] = {{0, 3, *INF*, 5},

                        {2, 0, *INF*, 4},

                        {*INF*, 1, 0, *INF*},

                        {*INF*, *INF*, 2, 0}};

*floydWarshall*(graph);

}

Output

