N QUEUES:

#include <stdio.h>

int n;

int check(int i,int j,int a[n][n]){

for(int r=0;r<n;++r)

if(a[i][r]==1) return 0;

for(int r=0;r<n;++r)

if(a[r][j]==1) return 0;

int x=i,y=j;

while(x<n && y<n){

if(a[x][y]==1) return 0;++x;++y;

}

x=i;y=j;

while(x>=0 && y>=0){

if(a[x][y]==1) return 0;--x;--y;}

x=i;y=j;

while(x>=0 && y<n){

if(a[x][y]==1) return 0;--x;++y;}

x=i;y=j;

while(x<n && y>=0){

if(a[x][y]==1) return 0;++x;--y;}

return 1;

}

void print(int a[n][n]){

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

for(int r=0;r<n;++r)

printf("%d ",a[i][r]);

printf("\n");

}

printf("\n\n");

}

void fn(int i,int a[n][n]){

if(i==n){

print(a);return;

}

for(int r=0;r<n;++r)

if(check(i,r,a)==1){

a[i][r]=1;

fn(i+1,a);

a[i][r]=0;

}

}

int main() {

printf("Enter the Number of queens : ");

scanf("%d",&n);

int a[n][n];

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

for(int r=0;r<n;++r) a[i][r]=0;

fn(0,a);

}

**Dijistra:**

#include <stdio.h>

#include <conio.h>

void main() {

int i, j, n, v, k, min, u, c[20][20], s[20], d[20];

clrscr();

printf("\n Enter the no. of vertices : ");+

scanf("%d", &n);

printf("\n Enter the cost adjacency matrix : ");

printf("\n Enter 999 for no edge ");

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

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

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

}

}

printf("\n Enter the source vertex : ");

scanf("%d", &v);

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

s[i] = 0;

d[i] = c[v][i];

}

d[v] = 0;

s[v] = 1;

for (k = 2; k <= n; k++) {

min = 999;

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

if ((s[i] == 0) && (d[i] < min)) {

min = d[i];

u = i;

}

}

s[u] = 1;

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

if (s[i] == 0) {

if (d[i] > (d[u] + c[u][i])) {

d[i] = d[u] + c[u][i];

}

}

}

}

printf("\n The shortest distance from %d is ", v);+

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

printf("\n %d --> %d = %d ", v, i, d[i]);

}

getch();

}

**Prims**

#include <stdio.h>

int cost[10][10], n, t[10][2], sum;

void prims(int cost[10][10], int n);

int main() {

int i, j;

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n");

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

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

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

}

}

prims(cost, n);

printf("Edges of the minimal spanning tree:\n");

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

printf("(%d, %d) ", t[i][0], t[i][1]);

}

printf("\nSum of minimal spanning tree: %d\n", sum);

return 0;

}

void prims(int cost[10][10], int n) {

int i, j, u, v;

int min, source;

int p[10], d[10], s[10];

min = 999;

source = 0;

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

d[i] = cost[source][i];

s[i] = 0;

p[i] = source;

}

s[source] = 1;

sum = 0;

int k = 0;

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

min = 999;

u = -1;

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

if (s[j] == 0 && d[j] < min) {

min = d[j];

u = j;

}

}

if (u != -1) {

t[k][0] = u;

t[k][1] = p[u];

k++;

sum += cost[u][p[u]];

s[u] = 1;

for (v = 0; v < n; v++) {

if (s[v] == 0 && cost[u][v] < d[v]) {

d[v] = cost[u][v];

p[v] = u;

}

}

}

}

}

**Kurskals**

#include <stdio.h>

int cost[10][10], n, t[10][2], sum;

void kruskal(int cost[10][10], int n);

int find(int parent[10], int i);

int main() {

int i, j;

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n");

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

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

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

}

}

kruskal(cost, n);

printf("Edges of the minimal spanning tree:\n");

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

printf("(%d, %d) ", t[i][0], t[i][1]);

}

printf("\nSum of minimal spanning tree: %d\n", sum);

return 0;

}

void kruskal(int cost[10][10], int n) {

int min, u, v, count, k;

int parent[10];

k = 0;

sum = 0;

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

parent[i] = i;

}

count = 0;

while (count < n - 1) {

min = 999;

u = -1;

v = -1;

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

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

if (find(parent, i) != find(parent, j) && cost[i][j] < min) {

min = cost[i][j];

u = i;

v = j;

}

}

}

int root\_u = find(parent, u);

int root\_v = find(parent, v);

if (root\_u != root\_v) {

parent[root\_u] = root\_v;

t[k][0] = u;

t[k][1] = v;

sum += min;

k++;

count++;

}

}

}

int find(int parent[10], int i) {

while (parent[i] != i) {

i = parent[i];

}

return i;

}

**Topological sorting:**

#include <stdio.h>

int n, a[10][10], res[10], s[10], top = 0;

void dfs(int, int, int[][10]);

void dfs\_top(int, int[][10]);

int main() {

printf("Enter the no. of nodes: ");

scanf("%d", &n);

int i, j;

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

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

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

}

}

dfs\_top(n, a);

printf("Solution: ");

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

printf("%d ", res[i]);

}

return 0;

}

void dfs\_top(int n, int a[][10]) {

int i;

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

s[i] = 0;

}

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

if (s[i] == 0) {

dfs(i, n, a);

}

}

}

void dfs(int j, int n, int a[][10]) {

s[j] = 1;

int i;

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

if (a[j][i] == 1 && s[i] == 0) {

dfs(i, n, a);

}

}

res[top++] = j;

}

**HEAP SORT**

#include <stdio.h>

#include <conio.h>

#include <time.h>

void heapcom(int a[], int n) {

int i, j, k, item;

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

item = a[i];

j = i;

k = j / 2;

while (k != 0 && item > a[k]) {

a[j] = a[k];

j = k;

k = j / 2;

}

a[j] = item;

}

}

void adjust(int a[], int n) {

int item, i, j;

j = 1;

item = a[j];

i = 2 \* j;

while (i < n) {

if ((i + 1) < n) {

if (a[i] < a[i + 1])

i++;

}

if (item < a[i]) {

a[j] = a[i];

j = i;

i = 2 \* j;

}

else {

break;

}

}

a[j] = item;

}

void heapsort(int a[], int n) {

int i, temp;

delay(1000);

heapcom(a, n);

for (i = n; i >= 1; i--) {

temp = a[1];

a[1] = a[i];

a[i] = temp;

adjust(a, i);

}

}

void main() {

int i, n, a[20], ch = 1;

clock\_t start, end;

clrscr();

while (ch) {

printf("\n enter the number of elements to sort\n");

scanf("%d", &n);

printf("\n enter the elements to sort\n");

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

scanf("%d", &a[i]);

start = clock();

heapsort(a, n);

end = clock();

printf("\n the sorted list of elements is\n");

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

printf("%d\n", a[i]);

printf("\n Time taken is %lf CPU cycles\n", (end - start) / (double)CLK\_TCK);

printf("do u wish to run again (0/1)\n");

scanf("%d", &ch);

}

getch();

}

**PERmutation**

#include <stdio.h>

#include <stdlib.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void generatePermutations(int arr[], int start, int end) {

if (start == end) {

for (int i = 0; i <= end; i++) {

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

}

printf("\n");

} else {

for (int i = start; i <= end; i++) {

swap(&arr[start], &arr[i]);

generatePermutations(arr, start + 1, end);

swap(&arr[start], &arr[i]);

}

}

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int\* arr = (int\*)malloc(n \* sizeof(int));

printf("Enter the elements: ");

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

scanf("%d", &arr[i]);

}

generatePermutations(arr, 0, n - 1);

free(arr);

return 0;

}

**Knapsack(DYNAmic)**

#include <stdio.h>

int i, j, n, c;

int w[10], p[10], v[10][10];

int max(int a, int b) {

return (a > b) ? a : b;

}

void knapsack(int n, int w[10], int p[10], int c) {

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

for (j = 0; j <= c; j++) {

if (i == 0 || j == 0)

v[i][j] = 0;

else if (w[i] > j)

v[i][j] = v[i - 1][j];

else

v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i]] + p[i]);

}

}

printf("\nMaximum Profit is: %d\n", v[n][c]);

printf("\nDP Table:\n");

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

for (j = 0; j <= c; j++) {

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

}

printf("\n");

}

}

int main() {

printf("Enter the number of objects: ");

scanf("%d", &n);

printf("Enter the weights:\n");

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

scanf("%d", &w[i]);

}

printf("Enter the profits:\n");

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

scanf("%d", &p[i]);

}

printf("Enter the knapsack capacity: ");

scanf("%d", &c);

knapsack(n, w, p, c);

return 0;

}

**Floyd**

#include <stdio.h>

int a[10][10], D[10][10], n;

void floyd(int [][10], int);

int min(int, int);

int main() {

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the cost adjacency matrix (use 999 for no direct path):\n");

int i, j;

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

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

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

}

}

floyd(a, n);

printf("Distance Matrix:\n");

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

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

printf("%d ", D[i][j]);

}

printf("\n");

}

return 0;

}

void floyd(int a[][10], int n) {

int i, j, k;

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

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

D[i][j] = a[i][j];

}

}

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

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

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

if (D[i][k] + D[k][j] < D[i][j]) {

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

}

}

}

}

}

int min(int a, int b) {

return (a < b) ? a : b;

}

**https://github.com/Samhithagit/ADA\_LAB**

**Fraction**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

struct Item {

int index;

int value;

int weight;

float ratio;

};

// Function to compare two items by ratio (for qsort)

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

float r1 = ((struct Item\*)a)->ratio;

float r2 = ((struct Item\*)b)->ratio;

return (r1 < r2) - (r1 > r2); // Sort in descending order

}

int main() {

struct Item items[MAX];

int n, capacity;

printf("Enter the number of items (max %d): ", MAX);

scanf("%d", &n);

if (n <= 0 || n > MAX) {

printf("Invalid number of items.\n");

return 1;

}

printf("Enter knapsack capacity: ");

scanf("%d", &capacity);

if (capacity <= 0) {

printf("Invalid knapsack capacity.\n");

return 1;

}

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

printf("Enter value and weight for item %d: ", i + 1);

scanf("%d %d", &items[i].value, &items[i].weight);

if (items[i].weight <= 0) {

printf("Weight must be positive.\n");

return 1;

}

items[i].ratio = (float)items[i].value / items[i].weight;

items[i].index = i + 1;

}

// Sort items by value-to-weight ratio

qsort(items, n, sizeof(struct Item), compare);

float totalValue = 0.0;

int totalWeight = 0;

int remaining = capacity;

printf("\nSelected Items:\n");

printf("Item\tWeight Taken\tType\n");

for (int i = 0; i < n && remaining > 0; i++) {

if (items[i].weight <= remaining) {

remaining -= items[i].weight;

totalValue += items[i].value;

totalWeight += items[i].weight;

printf("%d\t%d\t\tFull\n", items[i].index, items[i].weight);

} else {

float fraction = (float)remaining / items[i].weight;

totalValue += items[i].value \* fraction;

totalWeight += remaining;

printf("%d\t%.2f\t\tFractional\n", items[i].index, (float)remaining);

remaining = 0;

}

}

printf("\nTotal value in knapsack: %.2f\n", totalValue);

printf("Total weight used: %d/%d\n", totalWeight, capacity);

return 0;

}

**Merge Code:**

#include <stdio.h>

#include <time.h>

#include <stdlib.h> /\* To recognize exit function when compiling with gcc \*/

void split(int[], int, int);

void combine(int[], int, int, int);

void main() {

int a[15000], n, i, j, ch, temp;

clock\_t start, end;

while(1) {

printf("\n1: For manual entry of N value and array elements");

printf("\n2: To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3: To exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch) {

case 1:

printf("\nEnter the number of elements: ");

scanf("%d", &n);

printf("\nEnter array elements: ");

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

scanf("%d", &a[i]);

}

start = clock();

split(a, 0, n - 1);

end = clock();

printf("\nSorted array is: ");

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

printf("%d\t", a[i]);

}

printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) / CLOCKS\_PER\_SEC));

break;

case 2:

n = 500;

while(n <= 14500) {

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

// a[i] = random(1000); // Using the correct way to populate the array

a[i] = n - i; // Reverse order to maximize sorting time

}

start = clock();

split(a, 0, n - 1);

// Dummy loop to create delay

for(j = 0; j < 500000; j++) {

temp = 38 / 600;

}

end = clock();

printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) / CLOCKS\_PER\_SEC));

n = n + 1000;

}

break;

case 3:

exit(0);

}

getchar(); // Consume the newline character left in the input buffer

}

}

void split(int a[], int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

split(a, low, mid); // Recursively sort the left half

split(a, mid + 1, high); // Recursively sort the right half

combine(a, low, mid, high); // Merge the two halves

}

}

void combine(int a[], int low, int mid, int high) {

int c[15000], i, j, k;

i = k = low;

j = mid + 1;

// Merging the two halves

while(i <= mid && j <= high) {

if(a[i] < a[j]) {

c[k] = a[i];

++k;

++i;

} else {

c[k] = a[j];

++k;

++j;

}

}

// Copying the remaining elements from the left half

if(i > mid) {

while(j <= high) {

c[k] = a[j];

++k;

++j;

}

}

// Copying the remaining elements from the right half

if(j > high) {

while(i <= mid) {

c[k] = a[i];

++k;

++i;

}

}

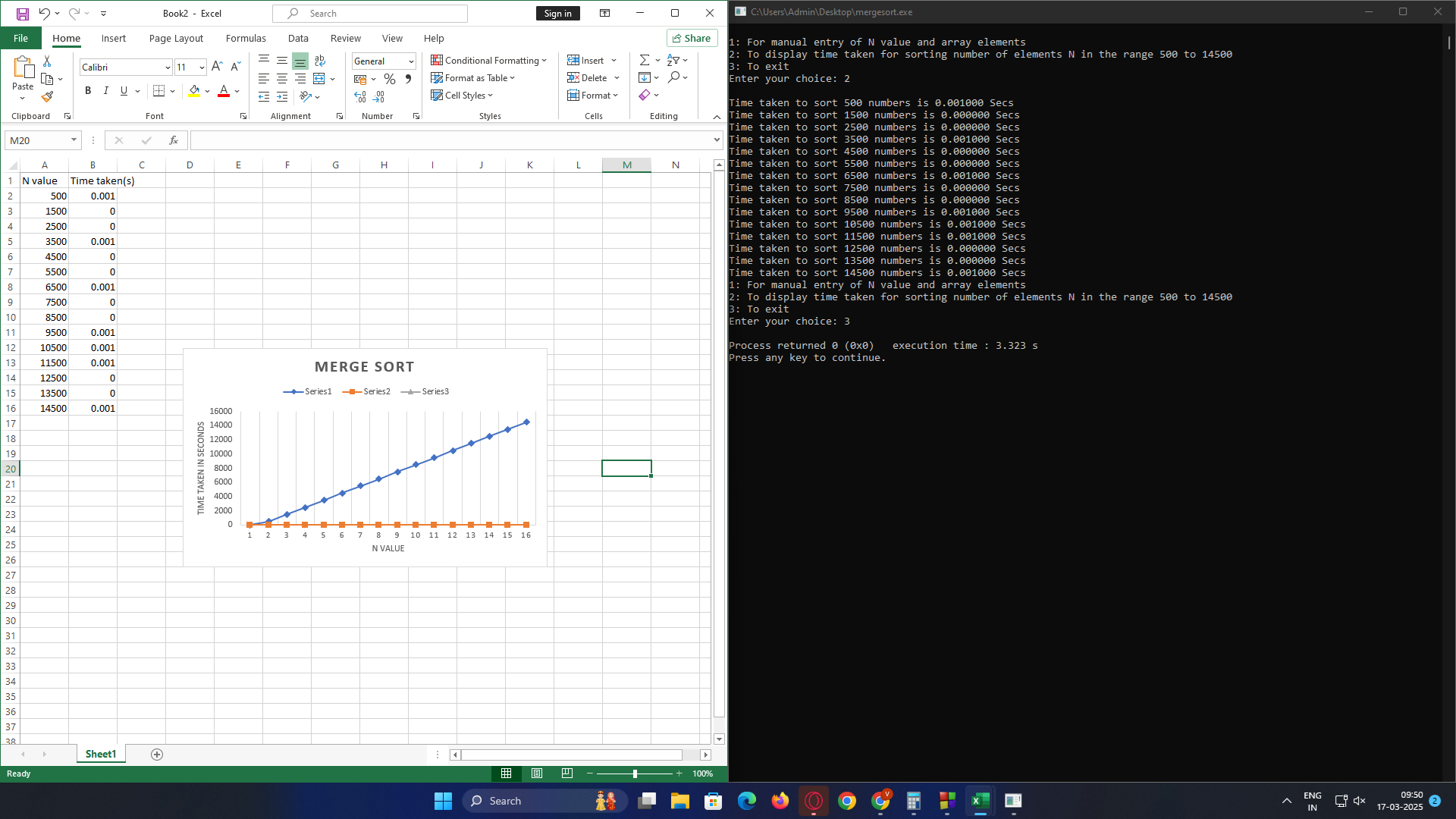
// Copying the merged array back to the original array

for(i = low; i <= high; i++) {

a[i] = c[i];

}

}

**OUTPUT:**

**Lab program 2:**

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX];

int visited[MAX];

int stack[MAX];

int top = -1;

void dfs(int v, int n) {

visited[v] = 1;

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

if (adj[v][i] && !visited[i]) {

dfs(i, n);

}

}

stack[++top] = v;

}

void topologicalSort(int n) {

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

visited[i] = 0;

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

if (!visited[i])

dfs(i, n);

printf("Topological Order (DFS): ");

while (top >= 0)

printf("%d ", stack[top--]);

printf("\n");

}

int main() {

int n, e, u, v;

printf("Enter number of vertices: ");

scanf("%d", &n);

printf("Enter number of edges: ");

scanf("%d", &e);

for (int i = 0; i < e; i++) {

printf("Enter edge (u v): ");

scanf("%d %d", &u, &v);

adj[u][v] = 1;

}

topologicalSort(n);

return 0;

}