EXP 1  
#include <stdio.h>

int main() {

int arr[100];

int n, i, j, minIndex, temp;

printf("Enter number of elements: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

}

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

minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

printf("Sorted array: ");

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

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

}

printf("\n");

return 0;

}

EXP2  
#include <stdio.h>

void insertionSort(int arr[], int n) {

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

int key = arr[i], j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

insertionSort(arr, n);

printf("Sorted array: ");

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

return 0;

}

Exp3

#include <stdio.h>

int binarysearch(int \*a, int low, int high, int key) {

if (low == high) {

if (a[low] == key)

return low;

else

return -1;

} else {

int mid = low + (high - low) / 2;

if (key == a[mid])

return mid;

else if (key > a[mid])

return binarysearch(a, mid + 1, high, key);

else

return binarysearch(a, low, mid - 1, key);

}

}

void main() {

int n, key;

printf("ENTER THE NUMBER OF ELEMENTS: ");

scanf("%d", &n);

int a[n];

printf("ENTER THE ELEMENTS: ");

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

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

int low = 0, high = n - 1;

printf("ENTER THE ELEMENT TO BE SEARCHED: ");

scanf("%d", &key);

int flag = binarysearch(a, low, high, key);

if (flag == -1)

printf("Search Unsuccessful\n");

else

printf("Search Successful, Element found at position: %d\n", flag);

}

EXP4  
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[l + i];

for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

int i = 0, j = 0, k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

mergeSort(arr, 0, n - 1);

printf("Sorted array: ");

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

return 0;

}

EXP5

#include<stdio.h>

#include<stdlib.h>

struct Item{

int weight;

int value;

float ratio;

};

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

struct Item\*item1=(struct Item\*)a;

struct Item\*item2=(struct Item\*)b;

if(item1->ratio<item2->ratio)

return 1;

else

return -1;

}

float fractionalKnapsack(int capacity,struct Item items[],int n) {

qsort(items,n,sizeof(items[0]),compare);

int currentWeight=0;

float totalValue=0.0;

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

if(currentWeight +items[i].weight<=capacity){

currentWeight +=items[i].weight;

totalValue+=items[i].value;

}else{

int remainingWeight=capacity-currentWeight;

totalValue+=items[i].value\*((float)remainingWeight/items[i].weight);

break;

}

}

return totalValue;

}

int main(){

int n,capacity;

printf("Enter number of items:");

scanf("%d",&n);

printf("Enter the capacity of Knapsack:");

scanf("%d",&capacity);

struct Item items[n];

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);

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

}

float maxValue=fractionalKnapsack(capacity,items,n);

printf("Maximum value in Knapsack=%.2f\n",maxValue);

return 0;

}

EXP 6  
#include <stdio.h>

#include <stdlib.h>

typedef struct {

int u, v, w;

} Edge;

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

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

}

int find(int p[], int x) {

return p[x] == x ? x : (p[x] = find(p, p[x]));

}

int main() {

int V, E;

scanf("%d %d", &V, &E);

Edge edges[E];

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

scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);

qsort(edges, E, sizeof(Edge), cmp);

int parent[V], rank[V];

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

parent[i] = i;

rank[i] = 0;

}

int mst\_wt = 0, count = 0;

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

int u = find(parent, edges[i].u);

int v = find(parent, edges[i].v);

if (u != v) {

mst\_wt += edges[i].w;

printf("%d-%d : %d\n", edges[i].u, edges[i].v, edges[i].w);

if (rank[u] < rank[v])

parent[u] = v;

else if (rank[u] > rank[v])

parent[v] = u;

else {

parent[v] = u;

rank[u]++;

}

count++;

}

}

printf("MST weight = %d\n", mst\_wt);

return 0;

}  
  
  
EXP 7  
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define INF 99999

void floydWarshall(int n, int dist[][n]) {

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

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

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

if (dist[i][k] + dist[k][j] < dist[i][j])

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

printf("\nShortest distances between every pair of vertices:\n");

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

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

if (dist[i][j] == INF)

printf("%5s", "INF");

else

printf("%5d", dist[i][j]);

}

printf("\n");

}

}

int main() {

int n;

printf("Enter number of vertices: ");

scanf("%d", &n);

int graph[n][n];

char temp[20];

printf("Enter adjacency matrix (use INF for no edge):\n");

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

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

scanf("%s", temp);

if (strcmp(temp, "INF") == 0)

graph[i][j] = INF;

else

graph[i][j] = atoi(temp);

}

}

floydWarshall(n, graph);

return 0;

}

EXP 8  
#include <stdio.h>

#include <string.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int main() {

char X[100], Y[100];

printf("Enter first string: ");

scanf("%s", X);

printf("Enter second string: ");

scanf("%s", Y);

int m = strlen(X), n = strlen(Y);

int dp[m + 1][n + 1];

// Build table dp[][]

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

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

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

dp[i][j] = 0;

else if (X[i - 1] == Y[j - 1])

dp[i][j] = 1 + dp[i - 1][j - 1];

else

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

// Length of LCS

int length = dp[m][n];

printf("Length of LCS: %d\n", length);

// Reconstruct LCS string

char lcs[length + 1];

lcs[length] = '\0';

int i = m, j = n, index = length - 1;

while (i > 0 && j > 0) {

if (X[i - 1] == Y[j - 1]) {

lcs[index--] = X[i - 1];

i--; j--;

} else if (dp[i - 1][j] > dp[i][j - 1])

i--;

else

j--;

}

printf("LCS: %s\n", lcs);

return 0;

}  
  
  
EXP 9  
#include <stdio.h>

void sumOfSubsets(int set[], int n, int target, int idx, int currSum, int subset[], int size) {

if (currSum == target) {

printf("{");

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

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

if (i < size - 1) printf(" ");

}

printf("}\n");

return;

}

if (idx >= n || currSum > target) return;

subset[size] = set[idx];

sumOfSubsets(set, n, target, idx + 1, currSum + set[idx], subset, size + 1); // include

sumOfSubsets(set, n, target, idx + 1, currSum, subset, size); // exclude

}

int main() {

int n, target;

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

scanf("%d", &n);

int set[n], subset[n];

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

for (int i = 0; i < n; i++) scanf("%d", &set[i]);

printf("Enter the target sum: ");

scanf("%d", &target);

printf("Subsets with sum equal to %d:\n", target);

sumOfSubsets(set, n, target, 0, 0, subset, 0);

return 0;

}

EXP 10  
#include <stdio.h>

#include <string.h>

void naiveStringMatching(char \*text, char \*pattern) {

int n = strlen(text);

int m = strlen(pattern);

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

int j;

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

if (text[i + j] != pattern[j]) break;

}

if (j == m) printf("Pattern found at index %d\n", i);

}

}

int main() {

char text[100], pattern[100];

printf("Enter the text: ");

scanf("%s", text);

printf("Enter the pattern: ");

scanf("%s", pattern);

naiveStringMatching(text, pattern);

return 0;

}