Day 2 Programs:

1.Write a C Program to perform MST using greedy technique:

Program:

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define MAX\_VERTICES 20

int findMinKey(int key[], bool mstSet[], int V) {

int min = INT\_MAX, min\_index;

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

if (!mstSet[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[MAX\_VERTICES][MAX\_VERTICES], int V) {

int minCost = 0;

printf("Edge \tWeight\n");

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

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

minCost += graph[i][parent[i]];

}

printf("Minimum Cost: %d\n", minCost);

}

void primMST(int graph[MAX\_VERTICES][MAX\_VERTICES], int V) {

int parent[MAX\_VERTICES];

int key[MAX\_VERTICES];

bool mstSet[MAX\_VERTICES];

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

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

int u = findMinKey(key, mstSet, V);

mstSet[u] = true;

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

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

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph, V);

}

int main() {

int V;

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

scanf("%d", &V);

int graph[MAX\_VERTICES][MAX\_VERTICES];

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

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

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

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

}

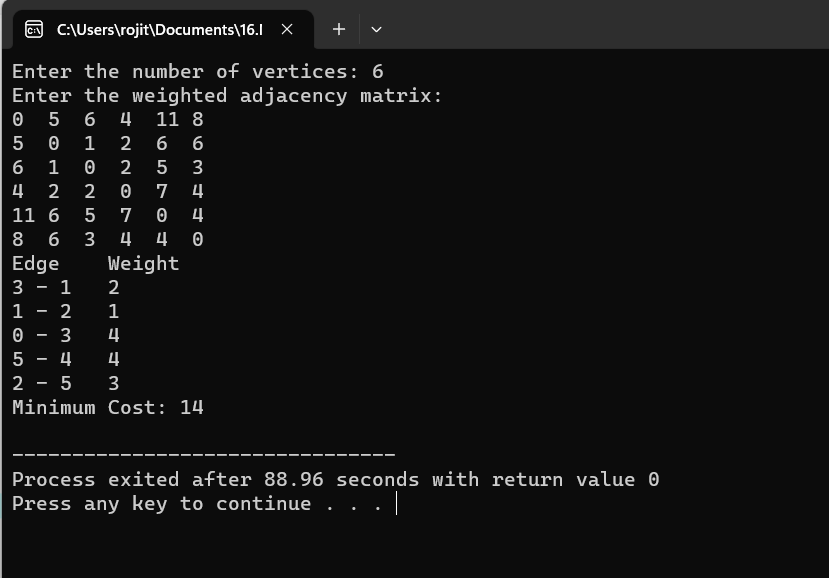
}

primMST(graph, V);

return 0;

}

Output:



2.Write a program to perform Knapsack Problem using Greedy algorithm:

Program:

#include <stdio.h>

#include <stdlib.h>

struct Item {

int value;

int weight;

float ratio;

};

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

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

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

return (itemB->ratio - itemA->ratio);

}

void sortByRatio(struct Item items[], int n) {

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

}

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

sortByRatio(items, n);

float totalValue = 0.0;

int currentWeight = 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].ratio \* remainingWeight;

break;

}

}

return totalValue;

}

int main() {

int n, capacity;

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

scanf("%d", &n);

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

scanf("%d", &capacity);

struct Item items[n];

printf("Enter the value and weight of each item:\n");

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

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

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

}

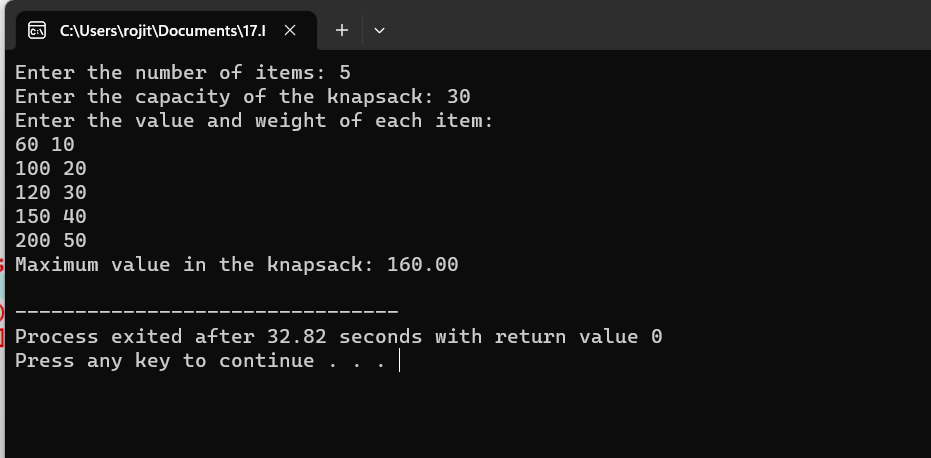
float maxValue = fractionalKnapsack(capacity, items, n);

printf("Maximum value in the knapsack: %.2f\n", maxValue);

return 0;

}

Output:



3.Write a Program using Dynamic programming concept to find out Optimal binary search tree:

Program:

#include <stdio.h>

#include <limits.h>

int optimalBST(int keys[], int freq[], int n) {

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

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

dp[i][i] = freq[i];

}

for (int L = 2; L <= n; L++) {

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

int j = i + L - 1;

dp[i][j] = INT\_MAX;

int sum = 0;

for (int k = i; k <= j; k++) {

sum += freq[k];

}

for (int k = i; k <= j; k++) {

int cost = ((k > i) ? dp[i][k - 1] : 0) +

((k < j) ? dp[k + 1][j] : 0) + sum;

if (cost < dp[i][j]) {

dp[i][j] = cost;

}

}

}

}

return dp[0][n - 1];

}

int main() {

int n;

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

scanf("%d", &n);

int keys[n], freq[n];

printf("Enter the keys: ");

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

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

}

printf("Enter the frequencies: ");

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

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

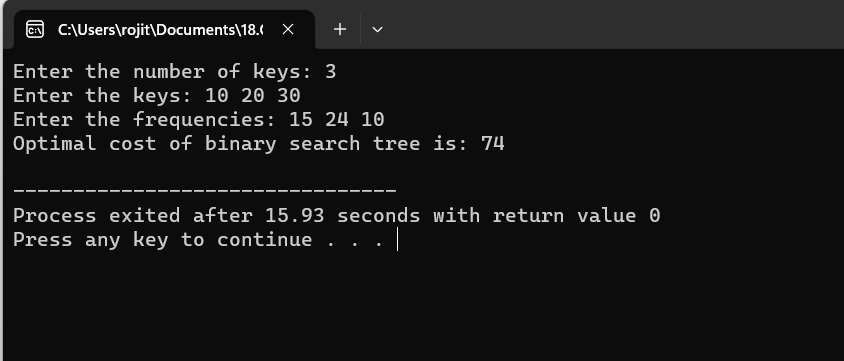
}

int optimalCost = optimalBST(keys, freq, n);

printf("Optimal cost of binary search tree is: %d\n", optimalCost);

}

Output:



19.Write a program using Dynamic Programming techniques to find Binomial Coefficient of given number:

Program:

#include <stdio.h>

unsigned long long binomialCoefficient(int n, int k) {

if (n < 0 || k < 0 || k > n) {

return 0;

}

unsigned long long C[n + 1][k + 1];

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

for (int j = 0; j <= i && j <= k; j++) {

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

C[i][j] = 1;

} else {

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

}

}

}

return C[n][k];

}

int main() {

int n, k;

printf("Enter the value of n: ");

scanf("%d", &n);

printf("Enter the value of k: ");

scanf("%d", &k);

unsigned long long coefficient = binomialCoefficient(n, k);

if (coefficient == 0) {

printf("Invalid input. Binomial coefficient is not defined for negative or invalid values.\n");

} else {

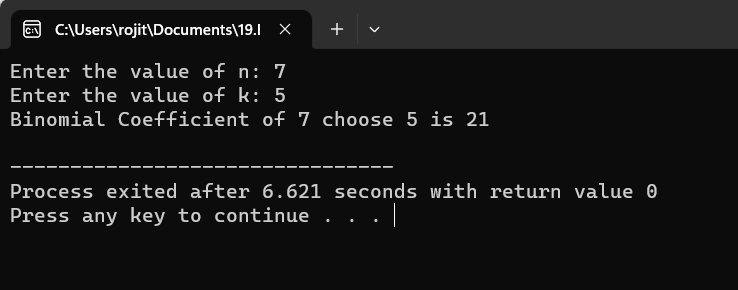
printf("Binomial Coefficient of %d choose %d is %llu\n", n, k, coefficient);

}

return 0;

}

Output:



20.Write a program to perform Merge Sort:

Program:

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) {

L[i] = arr[left + i];

}

for (int j = 0; j < n2; j++) {

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

}

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

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

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void printArray(int arr[], int size) {

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

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

}

printf("\n");

}

int main() {

int n;

printf("Enter the 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]);

}

printf("Original array: ");

printArray(arr, n);

mergeSort(arr, 0, n - 1);

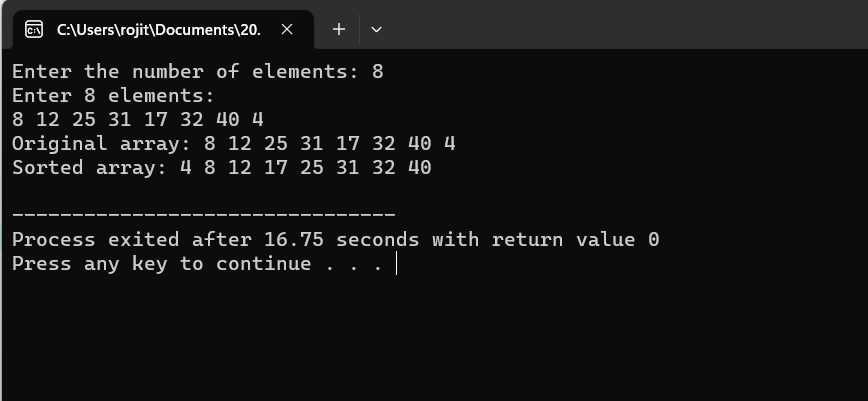
printf("Sorted array: ");

printArray(arr, n);

return 0;

}

Output:



21.Write a program using Divide and Conquer Strategy to fiond Min and Max value in the list:

Program:

#include <stdio.h>

struct MinMax {

int min;

int max;

};

struct MinMax findMinMax(int arr[], int left, int right) {

struct MinMax result, leftResult, rightResult;

if (left == right) {

result.min = arr[left];

result.max = arr[left];

return result;

}

if (left + 1 == right) {

if (arr[left] < arr[right]) {

result.min = arr[left];

result.max = arr[right];

} else {

result.min = arr[right];

result.max = arr[left];

}

return result;

}

int mid = (left + right) / 2;

leftResult = findMinMax(arr, left, mid);

rightResult = findMinMax(arr, mid + 1, right);

if (leftResult.min < rightResult.min) {

result.min = leftResult.min;

} else {

result.min = rightResult.min;

}

if (leftResult.max > rightResult.max) {

result.max = leftResult.max;

} else {

result.max = rightResult.max;

}

return result;

}

int main() {

int n,i;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

struct MinMax result = findMinMax(arr, 0, n - 1);

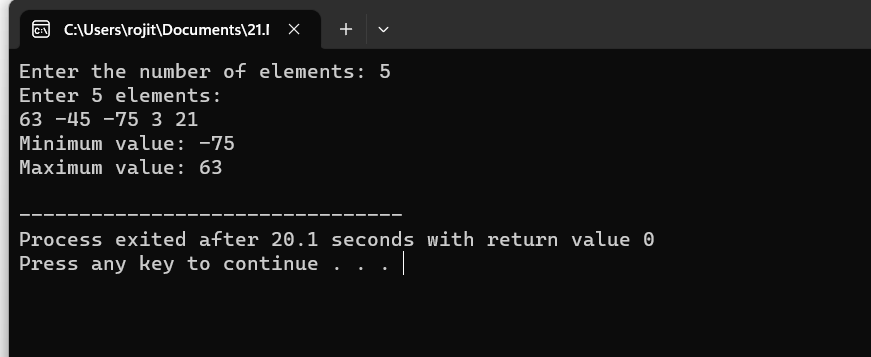
printf("Minimum value: %d\n", result.min);

printf("Maximum value: %d\n", result.max);

return 0;

}

Output:



22.Write a program to generate all the Prime numbers:

Program:

#include <stdio.h>

#include <stdbool.h>

bool isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

int main() {

int start, end;

printf("Enter the starting range: ");

scanf("%d", &start);

printf("Enter the ending range: ");

scanf("%d", &end);

printf("Prime numbers between %d and %d are:\n", start, end);

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

if (isPrime(num)) {

printf("%d ", num);

}

}

printf("\n");

return 0;

}

Output:

