**DAA PRACTICAL SESSION**

**Program 1:**

#include<stdio.h>

int fib(int n){

if(n==0)

return 0;

else if(n==1)

return 1;

else

return fib(n-1)+fib(n-2);

}

int main(){

int n=10;

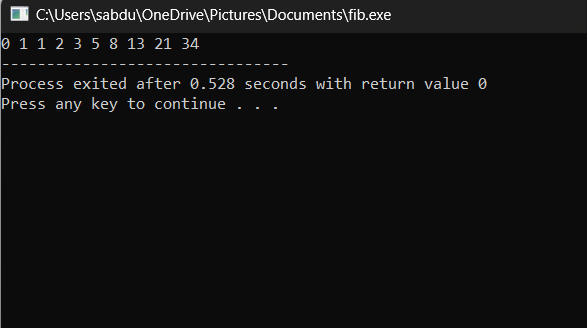
for(int i=0;i<n;i++){

printf("%d ",fib(i));

}

}

**Output** :

****

**Program 2:** Armstrong number

#include<stdio.h>

#include<math.h>

int main(){

int a,rem,c=0,s=0;

printf("enter the number : ");

scanf("%d",&a);

int t=a,m=a;

while(a!=0){

rem=a%10;

a/=10;

c++;}

while(t!=0){

rem=t%10;

s+=pow(rem,c);

t/=10;}

if(m==s)

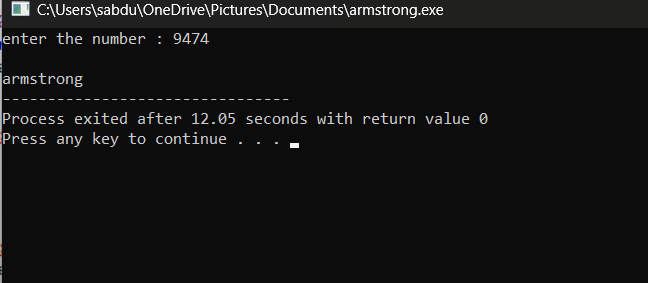
printf("\narmstrong");

else

printf("\n not armstrong");

}

**Output**:

****

**Program 3:**

#include<stdio.h>

int main(){

int a,b;

printf("enter the numbers: ");

scanf("%d %d",&a,&b);

while(a!=b){

if(a>b)

a=a-b;

else

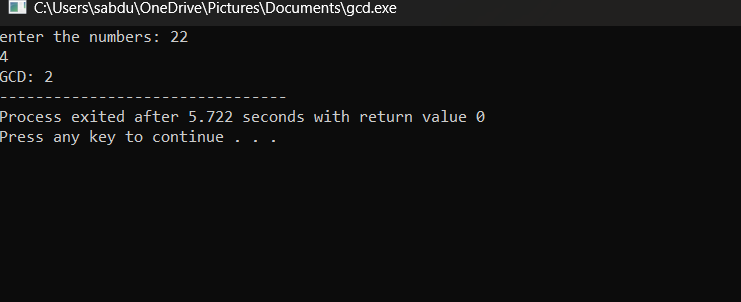
b=b-a;

}

printf("GCD: %d", a);

}

**Output**:



**Program 4:**

#include<stdio.h>

int main(){

int a[100],n;

printf("enter the no.of elememts ");

scanf("%d",&n);

printf("\nenter the elememts");

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

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

int k=a[0];

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

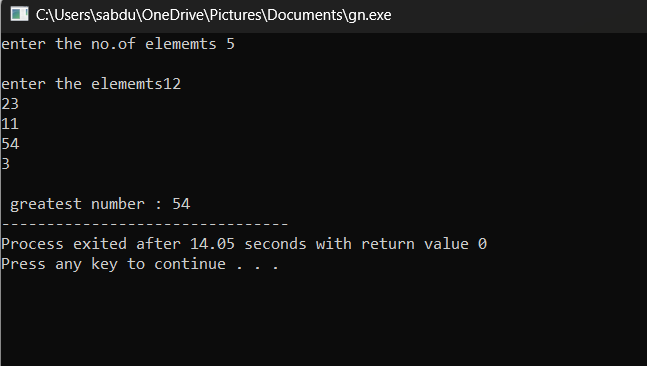
if(a[i]>k)

k=a[i];

printf("\n greatest number : %d",k);

}

**Output**:

****

**Program 5 :**

#include<stdio.h>

int main(){

int a,c=0;

printf("enter the number : ");

scanf("%d",&a);

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

if(a%i==0)

c++;

}

if(c<2)

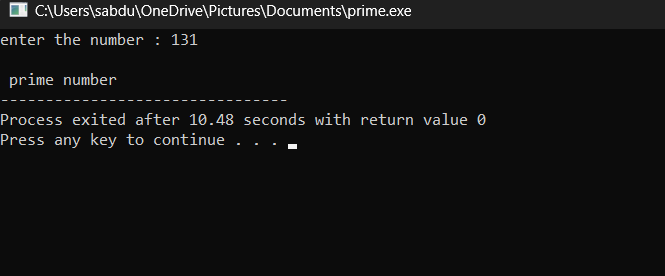
printf("\n prime number");

else

printf("\n not prime number");

}

**Output**:

****

**6. PRIME OR NOT**

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

int main() {

int i;

int num = 2;

bool isPrime = true;

if (num < 2) {

isPrime = false;

} else {

for (i = 2; i <= sqrt(num); i++) {

if (num % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime) {

printf("%d is prime.\n", num);

} else {

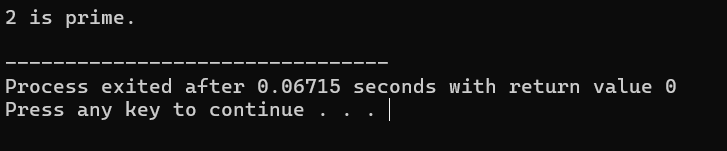
printf("%d is not prime.\n", num);

}

return 0;

}

**Output**:

****

**7.SELECTION SORT**

#include <stdio.h>

void selectionSort(int array[], int n) {

int i, j, min\_index, temp;

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

min\_index = i;

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

if (array[j] < array[min\_index]) {

min\_index = j;

}

}

if (min\_index != i) {

temp = array[i];

array[i] = array[min\_index];

array[min\_index] = temp;

}

}

}

void printArray(int array[], int n) {

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

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

}

printf("\n");

}

int main() {

int array[] = {64, 25, 12, 22, 11};

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

printf("Original array: \n");

printArray(array, n);

selectionSort(array, n);

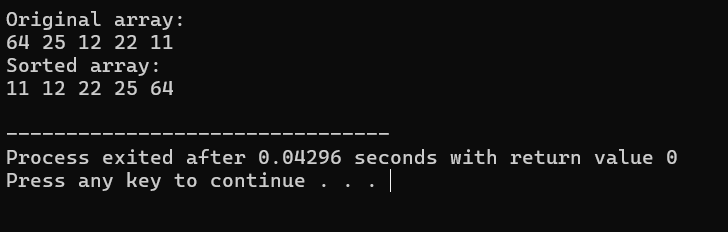
printf("Sorted array: \n");

printArray(array, n);

return 0;

}

**Output**:

****

**8.BUBBLE SORT**

#include <stdio.h>

void bubble\_sort(int a[], int length) {

int i, j, temp, flag;

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

flag = 0;

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

if (a[j] >a[j + 1]) {

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

flag = 1;

}

}

if (flag == 0)

break;

}

}

int main(void) {

int a[] = {3, 4, 9, 2, 1, 6};

int length = 6;

int i;

bubble\_sort(a, length);

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

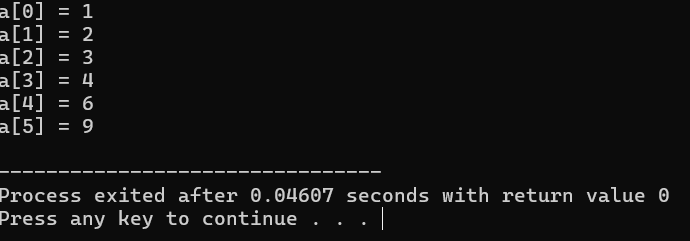
printf("a[%d] = %d\n", i, a[i]);

}

return 0;

}

**OUTPUT**:

****

**9.MULTIPLY TWO MATRICES**

#include <stdio.h>

int main() {

int a[2][2] = {{1, 2}, {3, 4}};

int b[2][2] = {{3, 4}, {2, 1}};

int c[2][2] = {{0, 0}, {0, 0}};

int i, j, k;

for (i = 0; i< 2; i++) {

for (j = 0; j < 2; j++) {

for (k = 0; k < 2; k++) {

c[i][j] += a[i][k] \* b[k][j];

}

}

}

for (i = 0; i< 2; i++) {

for (j = 0; j < 2; j++) {

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

}

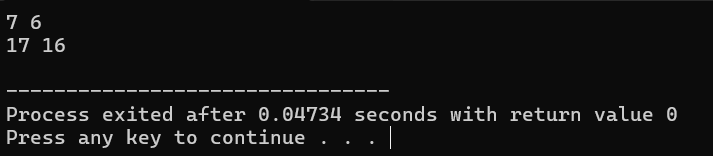
printf("\n");

}

return 0;

}

**OUTPUT**:



**10.PALINDROME**

#include <stdio.h>

#include <string.h>

int main() {

char str[100], reversed[100];

int len, i, is\_palindrome = 1;

printf("Enter the string: ");

scanf("%s", str);

len = strlen(str);

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

reversed[i] = str[len - i - 1];

}

reversed[len] = '\0';

if (strcmp(str, reversed) == 0) {

printf("Palindrome\n");

} else {

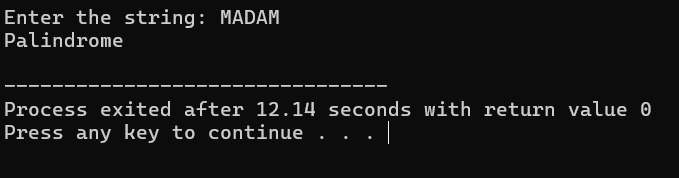
printf("Not a palindrome\n");

}

return 0;

}

**OUTPUT**:



**11.COPY ONE STRING TO ANOTHER**

#include <stdio.h> v

int main() {

char source[100], destination[100];

int i = 0;

printf("Enter a string: ");

fgets(source, sizeof(source), stdin);

while (source[i] != '\0') {

destination[i] = source[i];

i++;

}

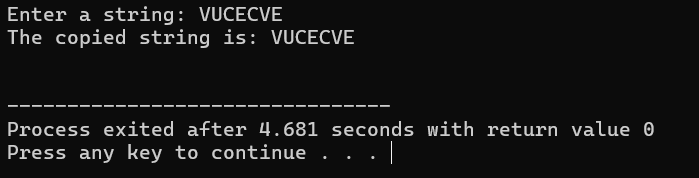
destination[i] = '\0';

printf("The copied string is: %s\n", destination);

return 0;

}

**OUTPUT:**



**12.BINARY SEARCH**

#include <stdio.h>

int binarySearch(int arr[], int size, int target) {

int low = 0, high = size - 1;

while (low <= high) {

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

if (arr[mid] == target) {

return mid;

}

if (arr[mid] < target) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() {

int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

int target, result;

printf("Enter the target value to search: ");

scanf("%d", &target);

result = binarySearch(arr, sizeof(arr) / sizeof(arr[0]), target);

if (result != -1) {

printf("Element found at index: %d\n", result);

} else {

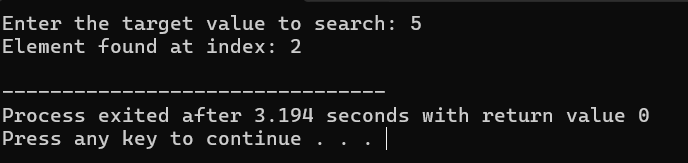
printf("Element not found\n");

}

return 0;

}

**OUTPUT**:



**13.REVERSE A STRING**

#include <stdio.h>

#include <string.h>

int main() {

char str[100], reversed[100];

int len, i;

strcpy(str, "vinay");

len = strlen(str);

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

reversed[i] = str[len - i - 1];

}

reversed[len] = '\0';

printf("%s\n", reversed);

return 0;

}

**WITHOUT USING FUNCTION**

#include <stdio.h>

int main() {

char str[] = "vinay";

char reversed[100];

int len = 0, i;

while (str[len] != '\0') {

len++;

}

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

reversed[i] = str[len - i - 1];

}

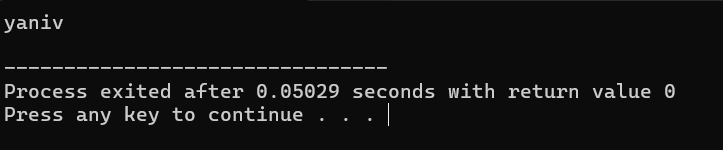
reversed[len] = '\0';

printf("%s\n", reversed);

return 0;

}

**OUTPUT**:



**14.LENGTH OF A STRING**#include <stdio.h>

int main() {

char str[100];

int length = 0;

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

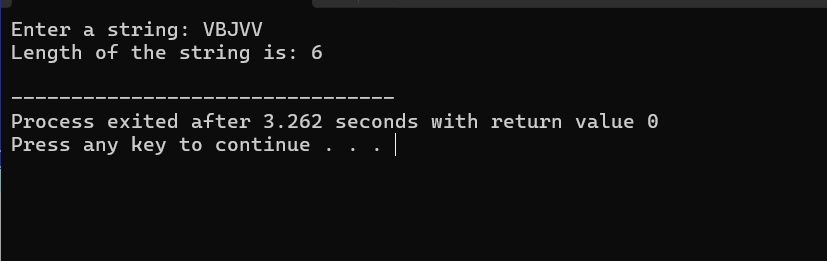
while (str[length] != '\0') {

length++;

}

printf("Length of the string is: %d\n", length);

return 0;

}  
**OUTPUT**: ****

**15. STRASSEN’S MULTIPLICATION**

#include <stdio.h>

#include <stdlib.h>

void addMatrix(int n, int A[n][n], int B[n][n], int result[n][n]) {

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

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

result[i][j] = A[i][j] + B[i][j];

}

}

}

void subtractMatrix(int n, int A[n][n], int B[n][n], int result[n][n]) {

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

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

result[i][j] = A[i][j] - B[i][j];

}

}

}

void strassenMultiply(int n, int A[n][n], int B[n][n], int C[n][n]) {

if (n == 1) {

C[0][0] = A[0][0] \* B[0][0];

return;

}

int newSize = n / 2;

int A11[newSize][newSize], A12[newSize][newSize], A21[newSize][newSize], A22[newSize][newSize];

int B11[newSize][newSize], B12[newSize][newSize], B21[newSize][newSize], B22[newSize][newSize];

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

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

A11[i][j] = A[i][j];

A12[i][j] = A[i][j + newSize];

A21[i][j] = A[i + newSize][j];

A22[i][j] = A[i + newSize][j + newSize];

B11[i][j] = B[i][j];

B12[i][j] = B[i][j + newSize];

B21[i][j] = B[i + newSize][j];

B22[i][j] = B[i + newSize][j + newSize];

}

}

int P1[newSize][newSize], P2[newSize][newSize], P3[newSize][newSize], P4[newSize][newSize];

int P5[newSize][newSize], P6[newSize][newSize], P7[newSize][newSize];

int temp1[newSize][newSize], temp2[newSize][newSize];

subtractMatrix(newSize, B12, B22, temp1);

strassenMultiply(newSize, A11, temp1, P1);

addMatrix(newSize, A11, A12, temp1);

strassenMultiply(newSize, temp1, B22, P2);

addMatrix(newSize, A21, A22, temp1);

strassenMultiply(newSize, temp1, B11, P3);

subtractMatrix(newSize, B21, B11, temp1);

strassenMultiply(newSize, A22, temp1, P4);

addMatrix(newSize, A11, A22, temp1);

addMatrix(newSize, B11, B22, temp2);

strassenMultiply(newSize, temp1, temp2, P5);

subtractMatrix(newSize, A12, A22, temp1);

addMatrix(newSize, B21, B22, temp2);

strassenMultiply(newSize, temp1, temp2, P6);

subtractMatrix(newSize, A11, A21, temp1);

addMatrix(newSize, B11, B12, temp2);

strassenMultiply(newSize, temp1, temp2, P7);

int C11[newSize][newSize], C12[newSize][newSize], C21[newSize][newSize], C22[newSize][newSize];

addMatrix(newSize, P5, P4, temp1);

subtractMatrix(newSize, temp1, P2, C11);

addMatrix(newSize, P1, P2, C12);

addMatrix(newSize, P3, P4, C21);

addMatrix(newSize, P1, P5, temp1);

subtractMatrix(newSize, temp1, P3, P7);

subtractMatrix(newSize, temp1, P7, C22);

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

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

C[i][j] = C11[i][j];

C[i][j + newSize] = C12[i][j];

C[i + newSize][j] = C21[i][j];

C[i + newSize][j + newSize] = C22[i][j];

}

}

}

int main() {

int n;

printf("Enter the size of the matrix (n x n): ");

scanf("%d", &n);

int A[n][n], B[n][n], C[n][n];

printf("Enter matrix A elements:\n");

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

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

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

}

}

printf("Enter matrix B elements:\n");

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

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

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

}

}

strassenMultiply(n, A, B, C);

printf("Product matrix C is:\n");

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

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

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

}

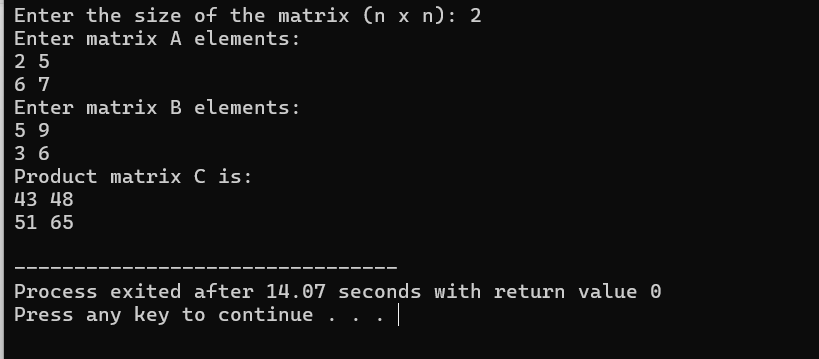
printf("\n");

}

return 0;

}

**OUTPUT**:



**16. MERGE SORT**

#include <stdio.h>

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

int n1 = mid - left + 1;

int n2 = right - mid;

int leftArr[n1], rightArr[n2];

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

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

}

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

rightArr[i] = arr[mid + 1 + i];

}

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

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

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

i++;

} else {

arr[k] = rightArr[j];

j++;

}

k++;

}

while (i< n1) {

arr[k] = leftArr[i];

i++;

k++;

}

while (j < n2) {

arr[k] = rightArr[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 arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is: \n");

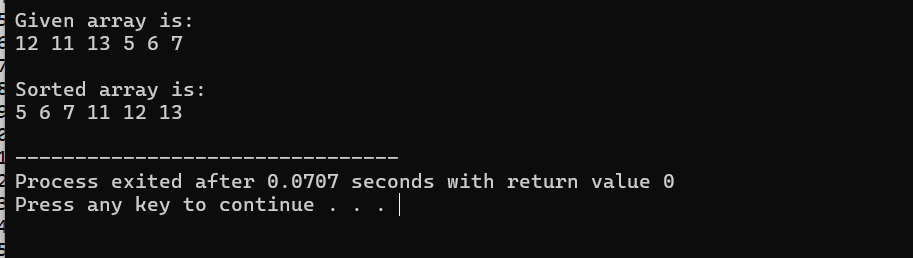
printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is: \n");

printArray(arr, arr\_size);

return 0;

}  
**OUTPUT**:

**17. MAX AND MIN IN THE LIST USING DIVIDE AND CONQUER METHOD**#include <stdio.h>

typedef struct {

int max;

int min;

} MaxMin;

MaxMinfindMaxMin(int arr[], int low, int high) {

MaxMin result, leftResult, rightResult;

if (low == high) {

result.max = arr[low];

result.min = arr[low];

return result;

}

int mid = (low + high) / 2;

leftResult = findMaxMin(arr, low, mid);

rightResult = findMaxMin(arr, mid + 1, high);

result.max = (leftResult.max>rightResult.max) ?leftResult.max : rightResult.max;

result.min = (leftResult.min<rightResult.min) ?leftResult.min : rightResult.min;

return result;

}

int main() {

int arr[] = {12, 5, 8, 20, 7, 15, 1};

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

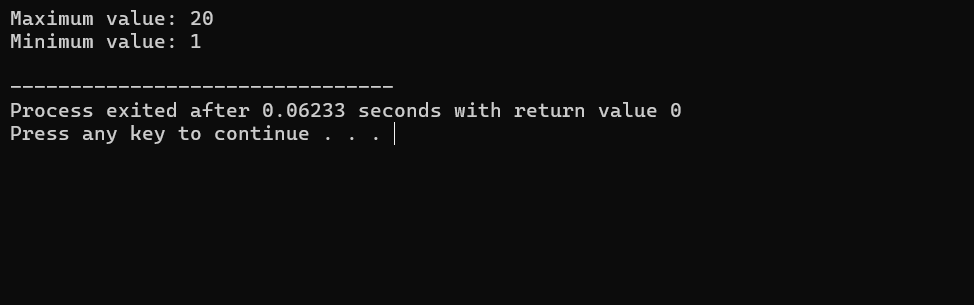
MaxMin result = findMaxMin(arr, 0, n - 1);

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

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

return 0;

}  
**OUTPUT**:



**18.PRIME NUMBERS BETWEEN 1 AND 100**

#include <stdio.h>

int isPrime(int num) {

if (num<= 1) {

return 0;

}

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

if (num % i == 0) {

return 0;

}

}

return 1;

}

int main() {

printf("Prime numbers between 1 and 100 are:\n");

for (int i = 1; i<= 100; i++) {

if (isPrime(i)) {

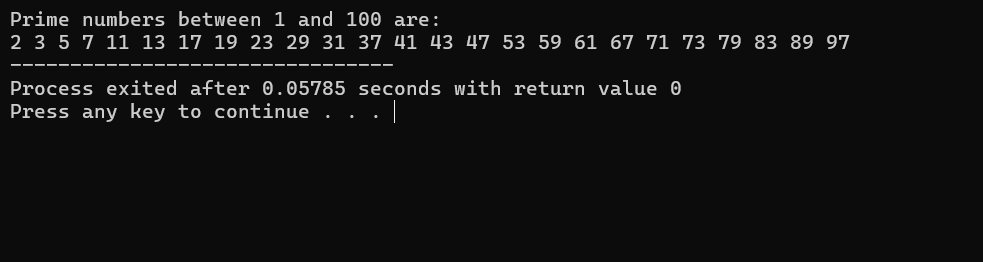
printf("%d ", i);

}

}

return 0;

}  
 **OUTPUT**:

****

**19.KNAPSACK PROBLEM USING GREEDY TECHNIQUES**#include <stdio.h>

#include <stdlib.h>

typedef struct {

int weight;

int value;

float ratio;

} Item;

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

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

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

return (item2->ratio > item1->ratio) - (item1->ratio > item2->ratio);

}

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

qsort(items, n, sizeof(Item), 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 the number of items: ");

scanf("%d", &n);

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

scanf("%d", &capacity);

Item items[n];

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

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

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

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

}

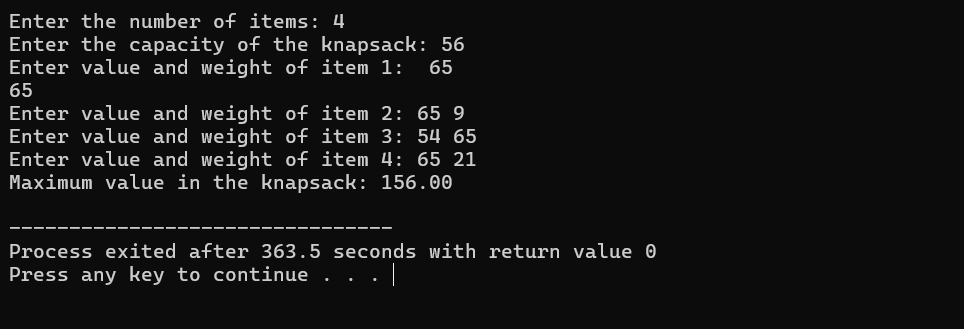
float maxValue = fractionalKnapsack(capacity, items, n);

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

return 0;

}

**OUTPUT**:

****

**20.MST USING GREEDY TECHNIQUE**#include <stdio.h>

#include <limits.h>

#define V 5

int minKey(int key[], int mstSet[]) {

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 primMST(int graph[V][V]) {

int parent[V], key[V], mstSet[V] = {0};

for (int i = 0; i< V; i++) key[i] = INT\_MAX;

key[0] = 0, parent[0] = -1;

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

int u = minKey(key, mstSet);

mstSet[u] = 1;

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];

}

printf("Edge \tWeight\n");

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

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

}

int main() {

int graph[V][V] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}

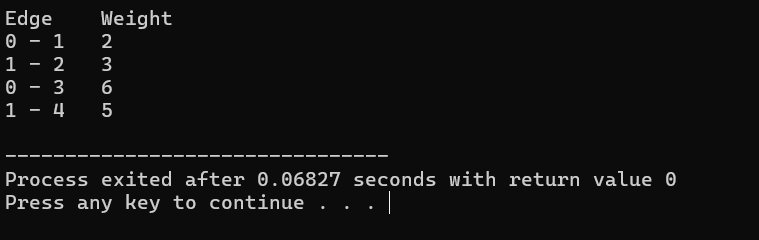
};

primMST(graph);

return 0;

}

**OUTPUT**:



**21.OBST USING DYNAMIC PROGRAMMING**#include <stdio.h>

#include <limits.h>

int sum(int freq[], int i, int j) {

int s = 0;

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

s += freq[k];

return s;

}

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

int cost[n][n];

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

cost[i][i] = freq[i];

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

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

int j = i + len - 1;

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

int fsum = sum(freq, i, j);

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

int c = ((r >i) ? cost[i][r - 1] : 0) +

((r < j) ? cost[r + 1][j] : 0) + fsum;

if (c < cost[i][j])

cost[i][j] = c;

}

}

}

return cost[0][n - 1];

}

int main() {

int keys[] = {10, 12, 20};

int freq[] = {34, 8, 50};

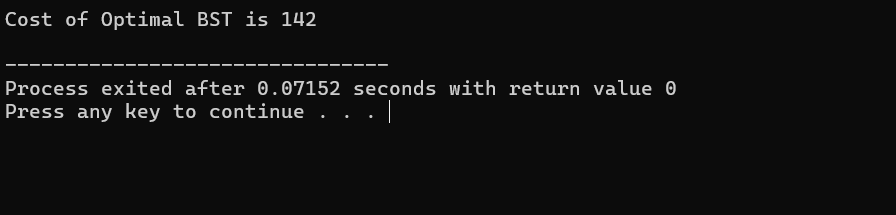
int n = sizeof(keys) / sizeof(keys[0]);

printf("Cost of Optimal BST is %d\n", optimalBST(keys, freq, n));

return 0;

}

**OUTPUT**:



**22.BINOMIAL COEFFICIENT USING DYNAMIC PROGRAMMING**#include <stdio.h>

int binomialCoeff(int n, int k) {

int C[n + 1][k + 1];

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

for (int j = 0; j <= (i<k ?i : 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() {

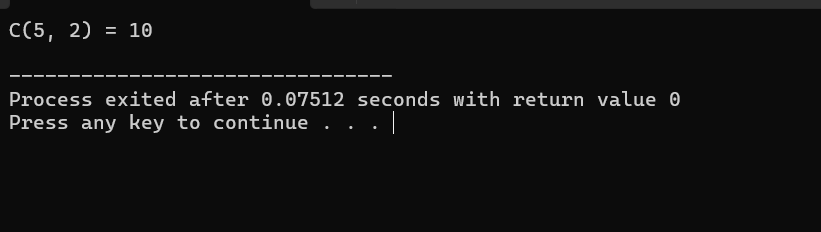
int n = 5, k = 2;

printf("C(%d, %d) = %d\n", n, k, binomialCoeff(n, k));

return 0;

}

**OUTPUT**:



**23.REVERSE A GIVEN NUMBER**#include <stdio.h>

int main() {

int num, reversed = 0;

printf("Enter a number: ");

scanf("%d", &num);

while (num != 0) {

reversed = reversed \* 10 + num % 10;

num /= 10;

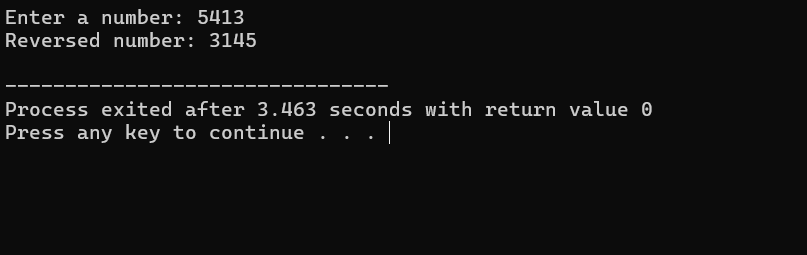
}

printf("Reversed number: %d\n", reversed);

return 0;

}

**OUTPUT**:



**24.PERFECT NUMBER**

#include <stdio.h>

int main() {

int num, sum = 0;

printf("Enter a number: ");

scanf("%d", &num);

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

if (num % i == 0)

sum += i;

}

if (sum == num)

printf("%d is a perfect number.\n", num);

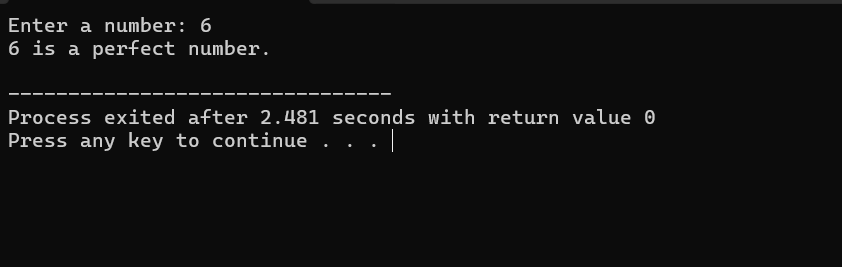
else

printf("%d is not a perfect number.\n", num);

return 0;

}

**OUTPUT**:

****

**25.TSP USING DYNAMIC PROGRAMMING**

#include <stdio.h>

#include <limits.h>

#define N 4

#define INF INT\_MAX

int dist[N][N] = {

{0, 20, 42, 35},

{20, 0, 30, 34},

{42, 30, 0, 12},

{35, 34, 12, 0}

};

int dp[1 << N][N];

int tsp(int mask, int pos) {

if (mask == ((1 << N) - 1))

return dist[pos][0];

if (dp[mask][pos] != -1)

return dp[mask][pos];

int ans = INF;

for (int city = 0; city < N; city++) {

if (!(mask & (1 << city))) {

int newAns = dist[pos][city] + tsp(mask | (1 << city), city);

if (newAns<ans)

ans = newAns;

}

}

return dp[mask][pos] = ans;

}

int main() {

for (int i = 0; i< (1 << N); i++)

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

dp[i][j] = -1;

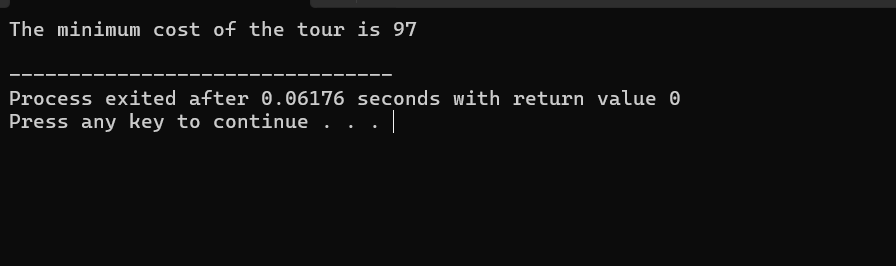
int result = tsp(1, 0);

printf("The minimum cost of the tour is %d\n", result);

return 0;

}

**OUTPUT**:

****

**26. PATTERN(RT)**

**1**

**1 2**

**1 2 3**

**1 2 3 4**#include <stdio.h>

int main() {

for (int i = 1; i<= 5; i++)

for (int j = 1; j <= i; j++)

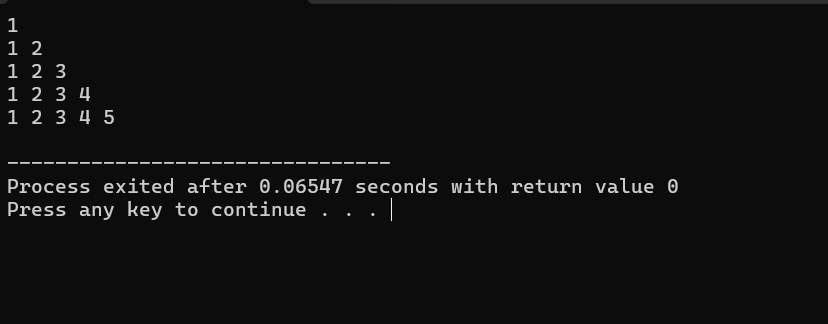
printf("%d ", j);

printf("\n");

return 0;

}

**OUTPUT**:

****

**27.FLOYD’S ALGORITHM**

#include <stdio.h>

#define INF 99999

#define V 4

void floydWarshall(int graph[V][V]) {

int dist[V][V], i, j, k;

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

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

dist[i][j] = graph[i][j];

}

}

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

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

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

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

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

}

}

}

}

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

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

if (dist[i][j] == INF) printf("INF ");

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

}

printf("\n");

}

}

int main() {

int graph[V][V] = {

{0, 3, INF, 7},

{8, 0, 2, INF},

{5, INF, 0, 1},

{2, INF, INF, 0}

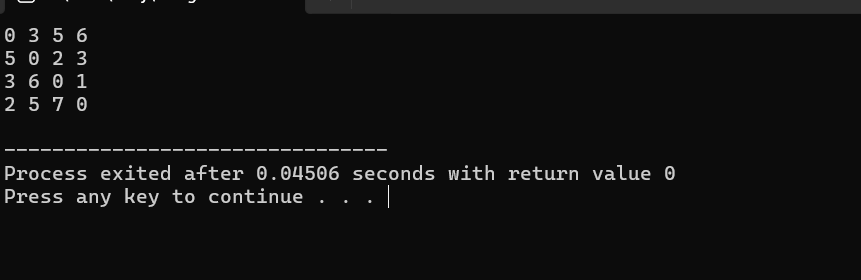
};

floydWarshall(graph);

return 0;

}

**OUTPUT**:

****

**28.PASCAL’S TRIANGLE**#include <stdio.h>

int main() {

int n, i, j, num;

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

scanf("%d", &n);

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

num = 1;

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

printf(" ");

}

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

printf("%d ", num);

num = num \* (i - j) / (j + 1);

}

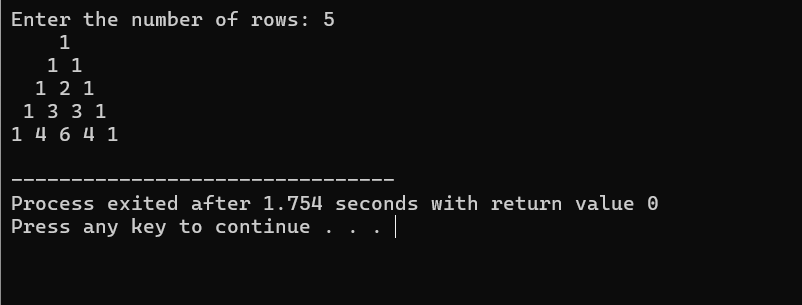
printf("\n");

}

return 0;

}

**OUTPUT**:

****

**29.SUM OF DIDGITS**#include <stdio.h>

int main() {

int num, sum = 0, digit;

printf("Enter a number: ");

scanf("%d", &num);

while (num != 0) {

digit = num % 10;

sum += digit;

num = num / 10;

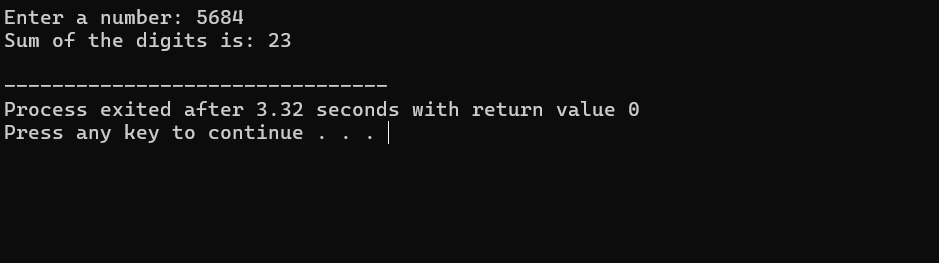
}

printf("Sum of the digits is: %d\n", sum);

return 0;

}

**OUTPUT**:

****

**30.INSERT A NUMBER IN THE LIST**#include <stdio.h>

int main() {

int arr[100], n, i, position, value;

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

scanf("%d", &n);

printf("Enter the elements of the array: \n");

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

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

}

printf("Enter the position to insert the number (1 to %d): ", n + 1);

scanf("%d", &position);

printf("Enter the value to insert: ");

scanf("%d", &value);

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

arr[i] = arr[i - 1];

}

arr[position - 1] = value;

n++;

printf("Updated array: ");

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

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

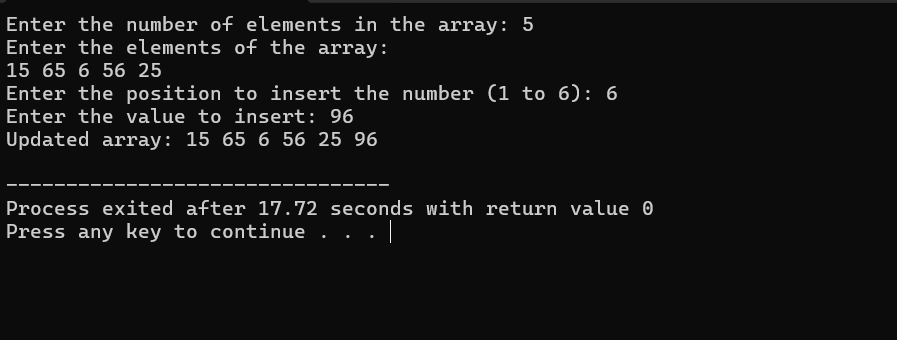
}

printf("\n");

return 0;

}

**OUTPUT**:

****

**31.SUM OF SUBSETS USING BACKTRACKING**#include <stdio.h>

void subsetSum(int arr[], int n, int target\_sum, int index, int current\_sum, int current\_subset[], int subset\_size) {

if (current\_sum == target\_sum) {

printf("{ ");

for (int i = 0; i<subset\_size; i++) {

printf("%d ", current\_subset[i]);

}

printf("}\n");

return;

}

if (current\_sum>target\_sum || index == n) {

return;

}

current\_subset[subset\_size] = arr[index];

subsetSum(arr, n, target\_sum, index + 1, current\_sum + arr[index], current\_subset, subset\_size + 1);

subsetSum(arr, n, target\_sum, index + 1, current\_sum, current\_subset, subset\_size);

}

void findAllSubsets(int arr[], int n, int target\_sum) {

int current\_subset[n];

subsetSum(arr, n, target\_sum, 0, 0, current\_subset, 0);

}

int main() {

int arr[] = {10, 7, 5, 18, 12, 20, 15};

int target\_sum = 35;

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

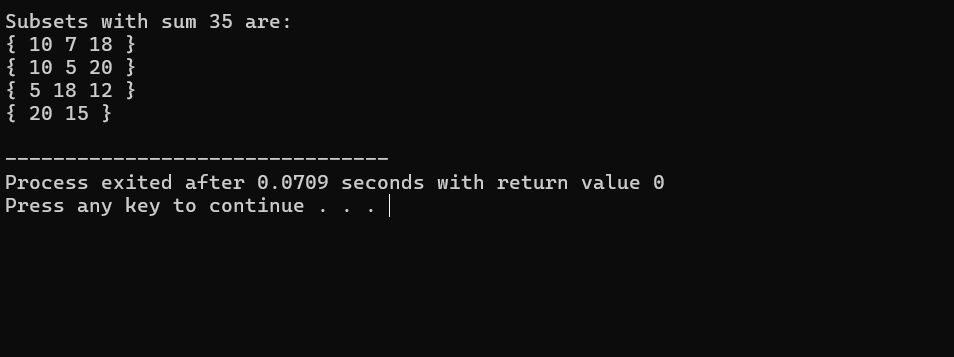
printf("Subsets with sum %d are:\n", target\_sum);

findAllSubsets(arr, n, target\_sum);

return 0;

}

**OUTPUT**:

****

**32.GRAPH COLOURING USING BACKTRACKING**#include <stdio.h>

#include <stdbool.h>

#define N 4

bool isSafe(int vertex, int graph[N][N], int colors[], int color) {

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

if (graph[vertex][i] &&colors[i] == color) {

return false;

}

}

return true;

}

bool graphColoring(int graph[N][N], int m, int colors[], int vertex) {

if (vertex == N) {

return true;

}

for (int color = 1; color<= m; color++) {

if (isSafe(vertex, graph, colors, color)) {

colors[vertex] = color;

if (graphColoring(graph, m, colors, vertex + 1)) {

return true;

}

colors[vertex] = 0;

}

}

return false;

}

void solveGraphColoring(int graph[N][N], int m) {

int colors[N] = {0};

if (graphColoring(graph, m, colors, 0)) {

printf("Solution found:\n");

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

printf("Vertex %d ->Color %d\n", i, colors[i]);

}

} else {

printf("No solution exists\n");

}

}

int main() {

int graph[N][N] = {

{0, 1, 1, 1},

{1, 0, 1, 0},

{1, 1, 0, 1},

{1, 0, 1, 0}

};

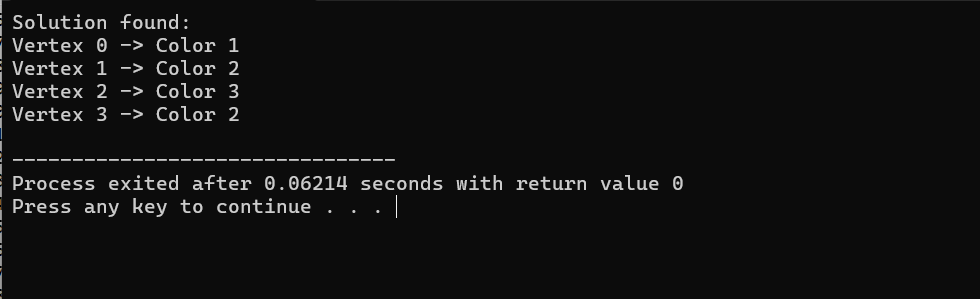
int m = 3;

solveGraphColoring(graph, m);

return 0;

}

**OUTPUT**:

****

**33.CONTAINER LOADING PROBLEM**

#include <stdio.h>

int maxLoad = 0;

void backtrack(int weights[], int n, int capacity, int index, int currentLoad) {

if (currentLoad> capacity) {

return;

}

if (currentLoad>maxLoad) {

maxLoad = currentLoad;

}

if (index == n) {

return;

}

backtrack(weights, n, capacity, index + 1, currentLoad + weights[index]);

backtrack(weights, n, capacity, index + 1, currentLoad);

}

int maxContainerLoad(int weights[], int n, int capacity) {

maxLoad = 0;

backtrack(weights, n, capacity, 0, 0);

return maxLoad;

}

int main() {

int weights[] = {10, 20, 30, 40};

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

int capacity = 50;

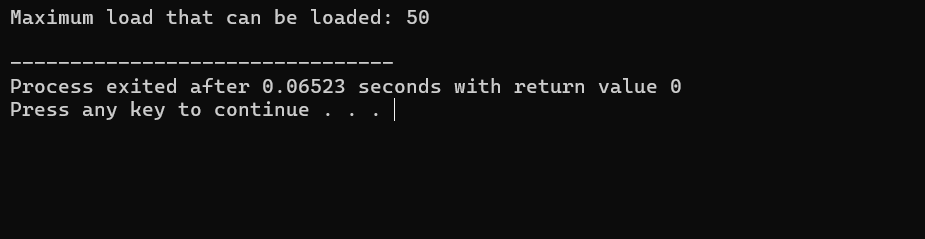
int maxLoadPossible = maxContainerLoad(weights, n, capacity);

printf("Maximum load that can be loaded: %d\n", maxLoadPossible);

return 0;

}

**OUTPUT**:

****

**34.LIST OF ALL FACTORS FOR N VALUE**#include <stdio.h>

#include <math.h>

void findFactors(int n) {

printf("Factors of %d are:\n", n);

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

if (n % i == 0) {

printf("%d ", i);

if (i != n / i) {

printf("%d ", n / i);

}

}

}

printf("\n");

}

int main() {

int n;

printf("Enter a number to find its factors: ");

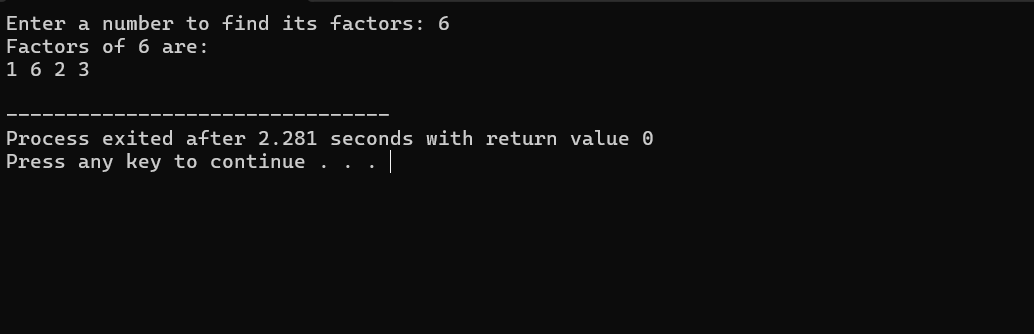
scanf("%d", &n);

findFactors(n);

return 0;

}

**OUTPUT**:

****

**35.JOB ASSIGNMENT PROBLEM USING BRANCH AND BOUND**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define N 4

typedef struct Node {

int cost;

int lowerBound;

int jobAssignment[N];

bool assigned[N];

int level;

} Node;

int calculateLowerBound(int costMatrix[N][N], bool assigned[N], int level) {

int lowerBound = 0;

for (int i = level; i< N; i++) {

int minCost = INT\_MAX;

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

if (!assigned[j] &&costMatrix[i][j] <minCost) {

minCost = costMatrix[i][j];

}

}

lowerBound += minCost;

}

return lowerBound;

}

void branchAndBound(int costMatrix[N][N]) {

int minCost = INT\_MAX;

Node bestNode;

Node root;

root.cost = 0;

root.level = 0;

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

root.assigned[i] = false;

root.jobAssignment[i] = -1;

}

root.lowerBound = calculateLowerBound(costMatrix, root.assigned, root.level);

Node queue[N \* N];

int queueSize = 0;

queue[queueSize++] = root;

while (queueSize> 0) {

Node currentNode = queue[--queueSize];

if (currentNode.lowerBound>= minCost) continue;

if (currentNode.level == N) {

if (currentNode.cost<minCost) {

minCost = currentNode.cost;

bestNode = currentNode;

}

continue;

}

for (int job = 0; job < N; job++) {

if (!currentNode.assigned[job]) {

Node newNode = currentNode;

newNode.level++;

newNode.jobAssignment[currentNode.level - 1] = job;

newNode.cost += costMatrix[currentNode.level - 1][job];

newNode.assigned[job] = true;

newNode.lowerBound = newNode.cost + calculateLowerBound(costMatrix, newNode.assigned, newNode.level);

if (newNode.lowerBound<minCost) {

queue[queueSize++] = newNode;

}

}

}

}

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

printf("Job assignments:\n");

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

printf("Person %d -> Job %d\n", i, bestNode.jobAssignment[i]);

}

}

int main() {

int costMatrix[N][N] = {

{9, 2, 7, 8},

{6, 4, 3, 7},

{5, 8, 1, 8},

{7, 6, 9, 4}

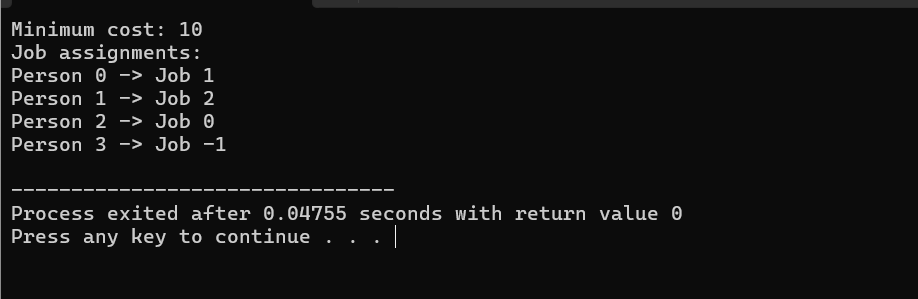
};

branchAndBound(costMatrix);

return 0;

}

**OUTPUT**:

****

**36.LINEAR SEARCH**#include <stdio.h>

int linearSearch(int arr[], int n, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int arr[] = {34, 21, 56, 78, 90, 23, 12};

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

int target = 78;

int result = linearSearch(arr, n, target);

if (result != -1) {

printf("Element found at index %d\n", result);

} else {

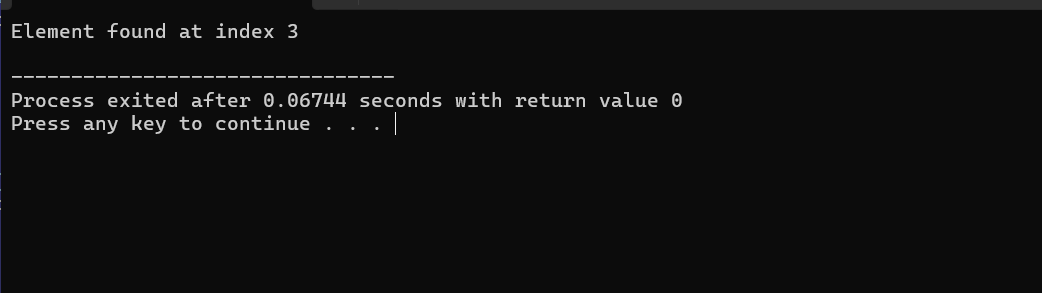
printf("Element not found in the array\n");

}

return 0;

}

**OUTPUT**:

****

**37.HAMILTONIAN CIRCUIT USING BACKTRACKING**#include <stdio.h>

#include <stdbool.h>

#define V 5

bool canAddToPath(int v, int graph[V][V], int path[], int position) {

if (graph[path[position - 1]][v] == 0)

return false;

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

if (path[i] == v)

return false;

}

return true;

}

bool hamiltonianCycle(int graph[V][V], int path[], int position) {

if (position == V) {

if (graph[path[position - 1]][path[0]] == 1)

return true;

else

return false;

}

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

if (canAddToPath(v, graph, path, position)) {

path[position] = v;

if (hamiltonianCycle(graph, path, position + 1))

return true;

path[position] = -1;

}

}

return false;

}

int main() {

int graph[V][V] = {

{0, 1, 0, 1, 0},

{1, 0, 1, 1, 0},

{0, 1, 0, 1, 1},

{1, 1, 1, 0, 1},

{0, 0, 1, 1, 0}

};

int path[V];

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

path[i] = -1;

}

path[0] = 0;

if (hamiltonianCycle(graph, path, 1)) {

printf("Hamiltonian Cycle found: \n");

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

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

}

printf("%d\n", path[0]);

} else {

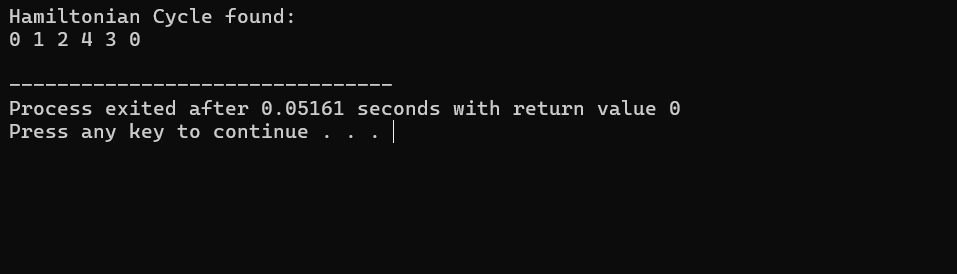
printf("No Hamiltonian Cycle found\n");

}

return 0;

}

**OUTPUT**:

****

**38.N QUEENS PROBLEM**

#include <stdio.h>

#include <stdbool.h>

#define N 8

int board[N][N];

void printSolution() {

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

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

if (board[i][j] == 1)

printf(" Q ");

else

printf(" . ");

}

printf("\n");

}

printf("\n");

}

bool isSafe(int row, int col) {

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

if (board[i][col] == 1)

return false;

}

for (int i = row, j = col; i>= 0 && j >= 0; i--, j--) {

if (board[i][j] == 1)

return false;

}

for (int i = row, j = col; i>= 0 && j < N; i--, j++) {

if (board[i][j] == 1)

return false;

}

return true;

}

bool solveNQueens(int row) {

if (row == N)

return true;

for (int col = 0; col < N; col++) {

if (isSafe(row, col)) {

board[row][col] = 1;

if (solveNQueens(row + 1))

return true;

board[row][col] = 0;

}

}

return false;

}

int main() {

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

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

board[i][j] = 0;

if (solveNQueens(0)) {

printSolution();

} else {

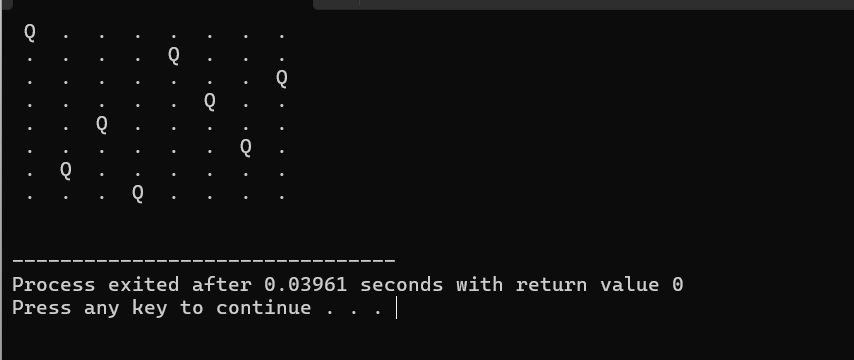
printf("No solution exists\n");

}

return 0;

}

**OUTPUT**:

****

**39.OPTIMAL COST BY USING APPROPRIATE ALGORITHM**#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5

#define INF INT\_MAX

void dijkstra(int graph[V][V], int src) {

int dist[V];

bool sptSet[V];

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

dist[i] = INF;

sptSet[i] = false;

}

dist[src] = 0;

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

int u = -1;

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

if (!sptSet[v] && (u == -1 || dist[v] <dist[u])) {

u = v;

}

}

sptSet[u] = true;

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

if (graph[u][v] && !sptSet[v] &&dist[u] != INF &&dist[u] + graph[u][v] <dist[v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

printf("Vertex\tDistance from Source\n");

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

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

}

}

int main() {

int graph[V][V] = {

{0, 10, 0, 30, 0},

{10, 0, 50, 0, 0},

{0, 50, 0, 20, 10},

{30, 0, 20, 0, 60},

{0, 0, 10, 60, 0}

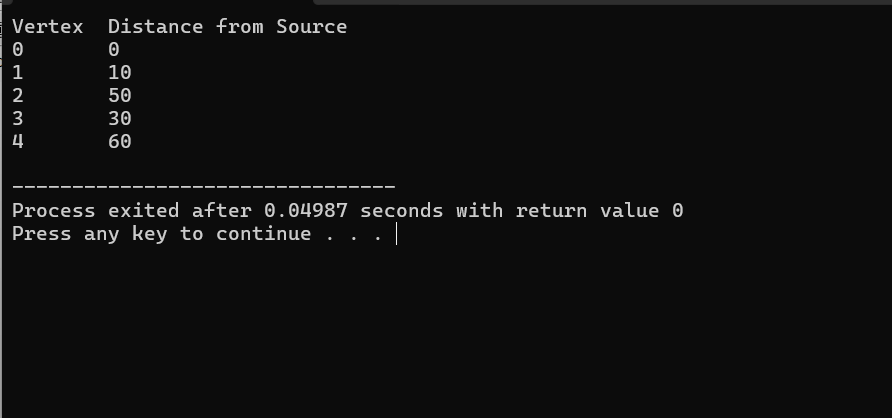
};

dijkstra(graph, 0);

return 0;

}

**OUTPUT**:

****

**40.MIN MAX VALUE SEPERATELY FOR ALL NUMBERS IN THE LIST**#include <stdio.h>

void findMinMax(int numbers[], int size, int\* min, int\* max) {

\*min = numbers[0];

\*max = numbers[0];

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

if (numbers[i] < \*min) {

\*min = numbers[i];

}

if (numbers[i] > \*max) {

\*max = numbers[i];

}

}

}

int main() {

int numbers[] = {34, 21, 56, 78, 90, 23, 12};

int size = sizeof(numbers) / sizeof(numbers[0]);

int min, max;

findMinMax(numbers, size, &min, &max);

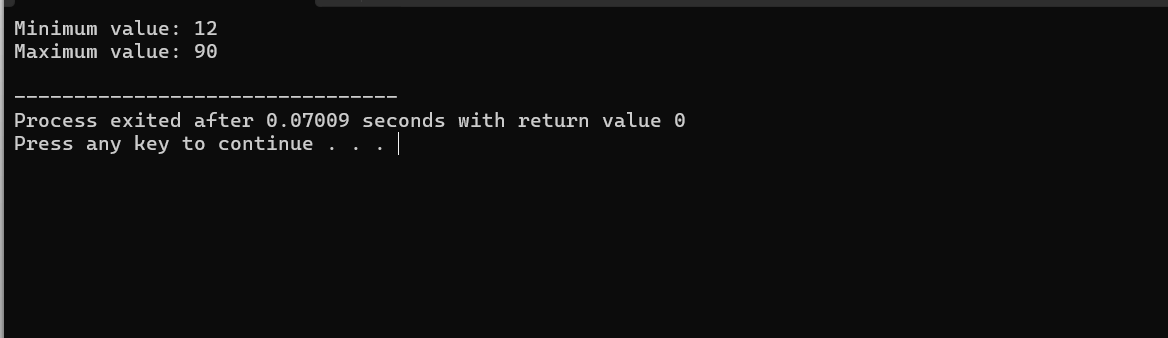
printf("Minimum value: %d\n", min);

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

return 0;

}

**OUTPUT**:

****