DAA Programs:

**1. write a program to print fibonacci series using recursion.**

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1)

return n;

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

}

int main() {

int n, i;

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

scanf("%d", &n);

printf("Fibonacci Series: ");

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

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

}

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of terms: 10

Sample Output:

Fibonacci Series: 0 1 1 2 3 5 8 13 21 34

**2.write a program to check the given number is armstrong or not .**

#include <stdio.h>

#include <math.h>

int isArmstrong(int num) {

int originalNum, remainder, n = 0, result = 0;

originalNum = num;

while (originalNum != 0) {

originalNum /= 10;

++n;

}

originalNum = num;

while (originalNum != 0) {

remainder = originalNum % 10;

result += pow(remainder, n);

originalNum /= 10;

}

return (result == num);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isArmstrong(num))

printf("%d is an Armstrong number.\n", num);

else

printf("%d is not an Armstrong number.\n", num);

return 0;

}

**Sample Input and Output:**

Sample Input 1:

Enter a number: 153

Sample Output 1:

153 is an Armstrong number.

**3.write a program to find the gcd of two numbers.**

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

int main() {

int num1, num2;

printf("Enter two integers: ");

scanf("%d %d", &num1, &num2);

printf("GCD of %d and %d is %d\n", num1, num2, gcd(num1, num2));

return 0;

}

**Sample Input and Output:**

Sample Input 1:

Enter two integers: 56 98

Sample Output 1:

GCD of 56 and 98 is 14.

**4.write a program to get the largest elements of array .**

#include <stdio.h>

int findLargest(int arr[], int n) {

int max = arr[0];

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

if (arr[i] > max)

max = arr[i];

}

return max;

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

printf("Largest element in the array is %d\n", findLargest(arr, n));

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of elements in the array: 5

Enter the elements of the array:

12 34 56 78 90

Sample Output:

The largest element in the array is 90.

**5.write a program to find the factorial of a niumber.**

#include <stdio.h>

int factorial(int n) {

if (n == 0)

return 1;

return n \* factorial(n - 1);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

printf("Factorial of %d is %d\n", num, factorial(num));

return 0;

}

**Sample Input and Output:**

Sample Input 1:

Enter a number: 5

Sample Output 1:

Factorial of 5 is 120.

**6.write a program to check a number is a prime number.**

#include <stdio.h>

int isPrime(int n) {

if (n <= 1)

return 0;

for (int i = 2; i <= n / 2; i++) {

if (n % i == 0)

return 0;

}

return 1;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isPrime(num))

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

else

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

return 0;

}

**Sample Input and Output:**

Sample Input 1:

Enter a number: 17

Sample Output 1:

17 is a prime number.

**7.write a program to perform selection sort .**

#include <stdio.h>

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

int i, j, minIndex, temp;

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;

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

selectionSort(arr, n);

printf("Sorted array: ");

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

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

}

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of elements in the array: 5

Enter the elements of the array:

64 25 12 22 11

**8.write a program to perform bubble sort.**

#include <stdio.h>

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

int i, j, temp;

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

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

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

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

} bubbleSort(arr, n);

printf("Sorted array: ");

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

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

}

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of elements in the array: 5

Enter the elements of the array:

64 34 25 12 22

Sample Output:

Sorted array:

12 22 25 34 64

**9.write a program to multiply to matrices .**

#include <stdio.h>

void multiplyMatrices(int firstMatrix[10][10], int secondMatrix[10][10], int result[10][10], int row1, int col1, int row2, int col2) {

for (int i = 0; i < row1; ++i) {

for (int j = 0; j < col2; ++j) {

result[i][j] = 0;

for (int k = 0; k < col1; ++k) {

result[i][j] += firstMatrix[i][k] \* secondMatrix[k][j];

}

}

}

}

int main() {

int firstMatrix[10][10], secondMatrix[10][10], result[10][10];

int row1, col1, row2, col2;

printf("Enter rows and columns for the first matrix: ");

scanf("%d %d", &row1, &col1);

printf("Enter rows and columns for the second matrix: ");

scanf("%d %d", &row2, &col2);

if (col1 != row2) {

printf("Matrix multiplication is not possible.\n");

return 0;

}

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

for (int i = 0; i < row1; ++i) {

for (int j = 0; j < col1; ++j) {

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

}

}

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

for (int i = 0; i < row2; ++i) {

for (int j = 0; j < col2; ++j) {

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

}

}

multiplyMatrices(firstMatrix, secondMatrix, result, row1, col1, row2, col2);

printf("Resultant Matrix:\n");

for (int i = 0; i < row1; ++i) {

for (int j = 0; j < col2; ++j) {

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

}

printf("\n");

}

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of rows and columns of the first matrix: 2 3

Enter the number of rows and columns of the second matrix: 3 2

Enter the elements of the first matrix:

1 2 3

4 5 6

Enter the elements of the second matrix:

7 8

9 10

11 12

Sample Output:

Resultant Matrix:

58 64

139 154

**10.write a program to check whether a given string as palindriome or not.**

#include <stdio.h>

#include <string.h>

int isPalindrome(char str[]) {

int len = strlen(str);

for (int i = 0; i < len / 2; i++) {

if (str[i] != str[len - i - 1]) {

return 0;

}

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

gets(str);

if (isPalindrome(str)) {

printf("%s is a palindrome.\n", str);

} else {

printf("%s is not a palindrome.\n", str);

}

return 0;

}

**Sample Input and Output:**

Sample Input 1:

Enter a string: racecar

Sample Output 1:

"racecar" is a palindrome.

**11.write a program to copy one string to another.**

#include <stdio.h>

#include <string.h>

int main() {

char str1[100], str2[100];

printf("Enter a string: ");

gets(str1);

strcpy(str2, str1);

printf("Copied string: %s\n", str2);

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the source string: Hello, World!

Sample Output:

Destination string: Hello, World!

**12.write a program to perform binary search.**

#include <stdio.h>

int binarySearch(int arr[], int n, int x) {

int left = 0, right = n - 1;

while (left <= right) {

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

if (arr[mid] == x)

return mid;

if (arr[mid] < x)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

int main() {

int arr[100], n, x;

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

scanf("%d", &n);

printf("Enter elements in sorted order: ");

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

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

printf("Enter the number to search: ");

scanf("%d", &x);

int result = binarySearch(arr, n, x);

if (result != -1)

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

else

printf("Element not found\n");

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of elements in the array: 10

Enter the elements of the sorted array:

2 4 6 8 10 12 14 16 18 20

Enter the target value to search: 14

Sample Output:

Element 14 found at index 6.

**13.write a program to print the reverse of a string.**

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

printf("Enter a string: ");

gets(str);

int len = strlen(str);

printf("Reversed string: ");

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

printf("%c", str[i]);

}

printf("\n");

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter a string: OpenAI

Sample Output:

Reversed string: IApneO

**14.write a program to find the length of a string .**

#include <stdio.h>

int main() {

char str[100];

int length = 0;

printf("Enter a string: ");

gets(str);

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

length++;

}

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

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter a string: Hello, World!

Sample Output:

Length of the string: 13

**15.write a program to perform strassens matrix multiplication.**

#include <stdio.h>

void strassenMultiply(int a[2][2], int b[2][2], int result[2][2]) {

int p1 = a[0][0] \* (b[0][1] - b[1][1]);

int p2 = (a[0][0] + a[0][1]) \* b[1][1];

int p3 = (a[1][0] + a[1][1]) \* b[0][0];

int p4 = a[1][1] \* (b[1][0] - b[0][0]);

int p5 = (a[0][0] + a[1][1]) \* (b[0][0] + b[1][1]);

int p6 = (a[0][1] - a[1][1]) \* (b[1][0] + b[1][1]);

int p7 = (a[0][0] - a[1][0]) \* (b[0][0] + b[0][1]);

result[0][0] = p5 + p4 - p2 + p6;

result[0][1] = p1 + p2;

result[1][0] = p3 + p4;

result[1][1] = p1 + p5 - p3 - p7;

}

int main() {

int a[2][2], b[2][2], result[2][2];

printf("Enter elements of 2x2 matrix A:\n");

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

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

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

printf("Enter elements of 2x2 matrix B:\n");

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

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

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

strassenMultiply(a, b, result);

printf("Resultant matrix:\n");

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

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

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

printf("\n");

}

return 0;

}

Sample Input and Output:

Sample Output:

Result matrix C:

19 22

43 50

**16.write a program to perform 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 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);

}

}

int main() {

int arr[100], n;

printf("Enter number of elements: ");

scanf("%d", &n);

printf("Enter elements: ");

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]);

}

printf("\n");

return 0;

}

**Sample Input and Output:**

Sample Output:

Given array:

12 11 13 5 6 7

Sorted array:

5 6 7 11 12 13

**17.using divide and conquer strategy to find max and min value in the list.**

#include <stdio.h>

struct Pair {

int min;

int max;

};

struct Pair findMinMax(int arr[], int low, int high) {

struct Pair minmax, mml, mmr;

int mid;

if (low == high) {

minmax.max = arr[low];

minmax.min = arr[low];

return minmax;

}

if (high == low + 1) {

if (arr[low] > arr[high]) {

minmax.max = arr[low];

minmax.min = arr[high];

} else {

minmax.max = arr[high];

minmax.min = arr[low];

}

return minmax;

}

mid = (low + high) / 2;

mml = findMinMax(arr, low, mid);

mmr = findMinMax(arr, mid + 1, high);

minmax.min = (mml.min < mmr.min) ? mml.min : mmr.min;

minmax.max = (mml.max > mmr.max) ? mml.max : mmr.max;

return minmax;

}

int main() {

int arr[] = {100, 300, 500, 2, 90, 800};

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

struct Pair minmax = findMinMax(arr, 0, n - 1);

printf("Minimum element is %d\n", minmax.min);

printf("Maximum element is %d\n", minmax.max);

return 0;

}

**Sample Input and Output:**

Sample Output:

Maximum value: 9

Minimum value: 1

**18.write a program to generate all the prime number(between 1 and 10).**

#include <stdio.h>

int isPrime(int n) {

if (n <= 1) return 0;

for (int i = 2; i <= n / 2; i++) {

if (n % i == 0)

return 0;

}

return 1;

}

int main() {

printf("Prime numbers between 1 and 10 are:\n");

for (int i = 1; i <= 10; i++) {

if (isPrime(i))

printf("%d ", i);

}

printf("\n");

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the upper limit to generate prime numbers: 30

Sample Output:

Prime numbers up to 30:

2 3 5 7 11 13 17 19 23 29

**19.write a program to perform knapsack problem using greedy techniques.**

#include <stdio.h>

struct Item {

int weight;

int value;

};

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

float ratio[n];

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

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

}

int totalValue = 0, currentWeight = 0;

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

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

currentWeight += items[i].weight;

totalValue += items[i].value;

}

}

printf("Maximum value in knapsack = %d\n", totalValue);

}

int main() {

struct Item items[] = {{60, 10}, {100, 20}, {120, 30}};

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

int capacity = 50;

knapsackGreedy(items, n, capacity);

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of items: 3

Item 1 - Value: 60

Item 1 - Weight: 10

Item 2 - Value: 100

Item 2 - Weight: 20

Item 3 - Value: 120

Item 3 - Weight: 30

Enter the capacity of the knapsack: 50

Sample Output:

Maximum value in the knapsack: 240.00

**20.write a program to perform MST using greedy techniques.**

#include <stdio.h>

#define MAX 100

#define INF 9999

int parent[MAX];

int find(int i) {

while (parent[i] != i)

i = parent[i];

return i;

}

void unionSets(int i, int j) {

int a = find(i);

int b = find(j);

parent[a] = b;

}

void kruskalMST(int n, int cost[MAX][MAX]) {

int mincost = 0;

printf("Edge \tWeight\n");

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

parent[i] = i;

int edges = 0;

while (edges < n - 1) {

int min = INF, a = -1, b = -1;

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

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

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

min = cost[i][j];

a = i;

b = j;

}

}

}

unionSets(a, b);

printf("%d - %d \t%d\n", a, b, min);

mincost += min;

edges++;

}

printf("Minimum cost = %d\n", mincost);

}

int main() {

int n;

int cost[MAX][MAX];

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

scanf("%d", &n);

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

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

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

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

if (cost[i][j] == 0)

cost[i][j] = INF;

}

}

kruskalMST(n, cost);

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the number of vertices: 4

Enter the number of edges: 5

Enter the edges (source, destination, weight):

Edge 1 - Source: 0

Edge 1 - Destination: 1

Edge 1 - Weight: 10

Edge 2 - Source: 0

Edge 2 - Destination: 2

Edge 2 - Weight: 6

Edge 3 - Source: 0

Edge 3 - Destination: 3

Edge 3 - Weight: 5

Edge 4 - Source: 1

Edge 4 - Destination: 3

Edge 4 - Weight: 15

Edge 5 - Source: 2

Edge 5 - Destination: 3

Edge 5 - Weight: 4

Sample Output:

Edges in the Minimum Spanning Tree:

0 -- 3 == 5

2 -- 3 == 4

0 -- 1 == 10

Minimum Cost: 19

**21.using dynamic programming concept to find out optimal binary search tree.**

#include <stdio.h>

#include <limits.h>

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 L = 2; L <= n; L++) {

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

int j = i + L - 1;

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

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

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

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

sum(freq, i, j);

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

cost[i][j] = c;

}

}

}

return cost[0][n - 1];

}

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

int s = 0;

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

s += freq[k];

return s;

}

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;

}

**Sample Input and Output:**

Sample Input:

Enter the number of keys: 4

Enter the probabilities of the keys:

Probability of key 1: 0.15

Probability of key 2: 0.10

Probability of key 3: 0.05

Probability of key 4: 0.20

Sample Output:

Minimum cost of the optimal BST: 1.10

Root table:

0 1 2 1

0 1 2 2

0 0 2 3

0 0 0 3

**22.using dynamic programming techniques to find binomial cofficient of a given number.**

#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("Binomial coefficient C(%d, %d) is %d\n", n, k, binomialCoeff(n, k));

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter the value of n: 5

Enter the value of k: 2

Sample Output:

C(5, 2) = 10

**23.write a program to find the reverse of a given number.**

#include <stdio.h>

int reverseNumber(int n) {

int rev = 0;

while (n != 0) {

rev = rev \* 10 + n % 10;

n /= 10;

}

return rev;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

printf("Reversed number is %d\n", reverseNumber(num));

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter a number: 12345

Sample Output:

Reversed number: 54321

**24.write a program to find the perfect number.**

#include <stdio.h>

int isPerfect(int n) {

int sum = 0;

for (int i = 1; i <= n / 2; i++) {

if (n % i == 0)

sum += i;

}

return (sum == n);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isPerfect(num))

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

else

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

return 0;

}

**Sample Input and Output:**

Sample Input:

Enter a number: 28

Sample Output:

28 is a perfect number.

**25.Write a program to perform travelling salesman problem using dynamic programming**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define MAX 16

#define INF INT\_MAX

int tsp(int n, int dist[MAX][MAX]) {

int dp[1 << MAX][MAX];

for (int mask = 0; mask < (1 << n); mask++) {

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

dp[mask][i] = INF;

}

}

dp[1][0] = 0;

for (int mask = 1; mask < (1 << n); mask += 2) {

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

if (!(mask & (1 << u))) continue;

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

if (mask & (1 << v)) continue;

int newMask = mask | (1 << v);

dp[newMask][v] = (dp[newMask][v] < dp[mask][u] +

dist[u][v]) ? dp[newMask][v] : dp[mask][u] + dist[u][v];

}

}

}

int answer = INF;

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

answer = (answer < dp[(1 << n) - 1][i] + dist[i][0]) ? answer :

dp[(1 << n) - 1][i] + dist[i][0];

}

return answer;

}

int main() {

int n;

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

scanf("%d", &n);

int dist[MAX][MAX];

printf("Enter the distance matrix:\n");

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

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

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

}

}

int result = tsp(n, dist);

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

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of cities: 4

Enter the distance matrix:

0 10 15 20

10 0 35 25

15 35 0 30

20 25 30 0

**Sample Output:**

The minimum cost of the TSP is: 80

**26.Write a program for the given pattern If n=4**

**1**

**1 2**

**1 2 3**

**1 2 3 4**

**C Code:**

**Here’s a C program to generate this pattern:**

#include <stdio.h>

void printPattern(int n) {

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

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

printf(" ");

}

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

printf("%d ", k);

}

printf("\n"); // Move to the next line

}

}

int main() {

int n;

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

scanf("%d", &n);

printPattern(n);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of rows (n): 4

**Sample Output:**

1

1 2

1 2 3

1 2 3 4

**27.Write a program to perform Floyd’s algorithm**

**C Code:**

**Here is a C program to perform Floyd-Warshall algorithm:**

#include <stdio.h>

#include <limits.h>

#define MAX 100

#define INF INT\_MAX

void floydWarshall(int graph[MAX][MAX], int n) {

int dist[MAX][MAX];

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

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

if (i == j) {

dist[i][j] = 0;

} else if (graph[i][j] != 0) {

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

} else {

dist[i][j] = INF;

}

}

}

// Floyd-Warshall algorithm

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] != INF && dist[k][j] != INF && dist[i][j] >

dist[i][k] + dist[k][j]) {

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

}

}

}

}

printf("Shortest 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("INF\t");

} else {

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

}

}

printf("\n");

}

}

int main() {

int n;

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

scanf("%d", &n);

int graph[MAX][MAX];

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

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

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

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

if (i != j && graph[i][j] == 0) {

graph[i][j] = INF; // Treat zero as infinity for non-

diagonal elements

}

}

}

floydWarshall(graph, n);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of vertices: 4

Enter the adjacency matrix:

0 3 0 7

8 0 2 0

5 0 0 1

2 0 0 0

**Sample Output:**

Shortest distances between every pair of vertices:

0 3 5 6

8 0 2 3

5 8 0 1

2 5 7 0

**28.Write a program for pascal triangle.**

**C Code:**

**Here is a C program to generate Pascal's Triangle for a given number of rows:**

#include <stdio.h>

void printPascalsTriangle(int n) {

int triangle[n][n];

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

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

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

triangle[i][j] = 1;

are 1

} else {

triangle[i][j] = triangle[i - 1][j - 1] + triangle[i -

1][j];

}

}

}

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

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

printf(" ");

}

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

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

}

printf("\n"); }

}

int main() {

int n;

printf("Enter the number of rows for Pascal's Triangle: ");

scanf("%d", &n);

printPascalsTriangle(n);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of rows for Pascal's Triangle: 5

**Sample Output:**

1

1 1

1 2 1

1 3 3 1

1. 4 6 4 1

**29.Write a program to find the optimal cost by using appropriate algorithm**

**C Code:**

**Here is a C program to solve the Knapsack Problem using dynamic programming:**

#include <stdio.h>

int knapsack(int W, int weights[], int values[], int n) {

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

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

for (int w = 0; w <= W; w++) {

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

dp[i][w] = 0;

} else if (weights[i - 1] <= w) {

dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]] >

dp[i - 1][w]) ?

(values[i - 1] + dp[i - 1][w - weights[i - 1]]):

dp[i - 1][w];

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][W];

}

int main() {

int n, W;

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

scanf("%d", &n);

int weights[n], values[n];

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

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

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

}

printf("Enter the values of the items:\n");

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

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

}

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

scanf("%d", &W);

int result = knapsack(W, weights, values, n);

printf("The maximum value that can be carried is: %d\n", result);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of items: 4

Enter the weights of the items:

2 3 4 5

Enter the values of the items:

3 4 5 6

Enter the maximum weight capacity of the knapsack: 5

**Sample Output:**

The maximum value that can be carried is: 7

**30. Write a program to find the sum of digits.**

**C Code:**

**Here's a C program to find the sum of digits of a given number:**

#include <stdio.h>

int sumOfDigits(int num) {

int sum = 0;

while (num != 0) {

sum += num % 10; num /= 10; }

return sum;

}

int main() {

int number;

printf("Enter a number: ");

scanf("%d", &number);

if (number < 0) {

number = -number;}

int result = sumOfDigits(number);

printf("The sum of digits is: %d\n", result);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter a number: 1234

**Sample Output:**

The sum of digits is: 10

**31.Write a program to print a minimum and maximum value sequency for all the numbers in**

**a list.**

**C Code:**

**Here's a C program that finds and prints the minimum and maximum values in a list of**

**numbers:**

#include <stdio.h>

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

\*min = arr[0];

\*max = arr[0];

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

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

\*min = arr[i];

}

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

\*max = arr[i];

}

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

int min, max;

findMinMax(arr, n, &min, &max);

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

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

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of elements: 5

Enter the elements:

3 1 4 1 5

**Sample Output:**

Minimum value: 1

Maximum value: 5

**32.Write a program to perform n Queens problem using backtracking.**

**C Code:**

**Here’s a C program to solve the N-Queens problem using backtracking:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 20

void printSolution(int board[MAX][MAX], int N) {

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

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

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

}

printf("\n");

}

printf("\n");

}

bool isSafe(int board[MAX][MAX], int row, int col, int N) {

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

if (board[i][col]) {

return false;

}

}

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

if (board[i][j]) {

return false;

}

}

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

if (board[i][j]) {

return false;

}

}

return true;

}

bool solveNQueens(int board[MAX][MAX], int row, int N) {

if (row >= N) {

return true;

}

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

if (isSafe(board, row, col, N)) {

board[row][col] = 1; // Place queen

if (solveNQueens(board, row + 1, N)) {

return true; }

board[row][col] = 0;

}

}

return false;

}

int main() {

int N;

int board[MAX][MAX] = {0};

printf("Enter the number of queens (N): ");

scanf("%d", &N);

if (solveNQueens(board, 0, N)) {

printf("One possible solution is:\n");

printSolution(board, N);

} else {

printf("No solution exists for N = %d\n", N);

}

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of queens (N): 4

**Sample Output:**

One possible solution is:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

**33. Write a program to inset a number in a list.**

**C Code:**

#include <stdio.h>

#define MAX 100

void insertNumber(int list[], int \*size, int number, int position) {

if (position < 0 || position > \*size) {

printf("Invalid position!\n");

return;

}

if (\*size >= MAX) {

printf("List is full!\n");

return;

}

for (int i = \*size; i > position; i--) {

list[i] = list[i - 1];

}

list[position] = number;

(\*size)++;

}

void printList(int list[], int size) {

printf("List elements are:\n");

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

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

}

printf("\n");

}

int main() {

int list[MAX];

int size = 0;

int number, position;

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

scanf("%d", &size);

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

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

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

}

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

scanf("%d", &number);

printf("Enter the position to insert the number at (0-based index): ");

scanf("%d", &position);

insertNumber(list, &size, number, position);

printList(list, size);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of initial elements in the list: 5

Enter the elements of the list:

10 20 30 40 50

Enter the number to insert: 25

Enter the position to insert the number at (0-based index): 2

**Sample Output:**

List elements are:

10 20 25 30 40 50

**34.Write a program to perform sum of subsets problem using backtracking**

**C Code:**

**Here’s a C program to solve the Sum of Subsets problem using backtracking:**

#include <stdio.h>

#define MAX 20

void printSubset(int subset[], int size) {

printf("{ ");

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

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

}

printf("}\n");

}

void findSubsets(int arr[], int n, int index, int target, int currentSum,

int subset[], int subsetSize) {

if (currentSum == target) {

printSubset(subset, subsetSize);

return;

}

if (index >= n || currentSum > target) {

return;

}

// Include the current element in the subset

subset[subsetSize] = arr[index];

findSubsets(arr, n, index + 1, target, currentSum + arr[index], subset,

subsetSize + 1);

findSubsets(arr, n, index + 1, target, currentSum, subset, subsetSize);

}

int main() {

int arr[MAX], n, target;

int subset[MAX];

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

scanf("%d", &n);

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

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

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

}

printf("Enter the target sum: ");

scanf("%d", &target);

printf("Subsets that sum up to %d are:\n", target);

findSubsets(arr, n, 0, target, 0, subset, 0);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of elements: 5

Enter the elements:

1 2 3 4 5

Enter the target sum: 5

**Sample Output:**

Subsets that sum up to 5 are:

{ 1 4 }

{ 2 3 }

{ 5 }

**35.Write a program to perform graph coloring problem using backtracking.**

**C Code:**

**Here's a C program that performs graph coloring using backtracking:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

bool isSafe(int graph[MAX\_VERTICES][MAX\_VERTICES], int color[], int v, int

c, int V) {

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

if (graph[v][i] && color[i] == c) {

return false;

}

}

return true;

}

bool graphColoringUtil(int graph[MAX\_VERTICES][MAX\_VERTICES], int color[],

int v, int m, int V) {

if (v == V) {

return true;

}

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

if (isSafe(graph, color, v, c, V)) {

color[v] = c;

if (graphColoringUtil(graph, color, v + 1, m, V)) {

return true;

}

remove it

color[v] = 0;

}

}

return false;

}

bool graphColoring(int graph[MAX\_VERTICES][MAX\_VERTICES], int V, int m) {

int color[V];

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

color[i] = 0;

}

return graphColoringUtil(graph, color, 0, m, V);

}

void printSolution(int color[], int V) {

printf("Solution:\n");

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

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

}

}

int main() {

int V, E, m;

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

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

scanf("%d", &V);

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

scanf("%d", &E);

printf("Enter the edges (format: u v):\n");

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

int u, v;

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

graph[u][v] = 1;

graph[v][u] = 1;

}

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

scanf("%d", &m);

if (graphColoring(graph, V, m)) {

printf("Solution exists with %d colors:\n", m);

printSolution(color, V);

} else {

printf("Solution does not exist with %d colors.\n", m);

}

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of vertices: 4

Enter the number of edges: 4

Enter the edges (format: u v):

0 1

0 2

1 2

1 3

Enter the number of colors: 3

**Sample Output:**

Solution exists with 3 colors:

Vertex 0 ---> Color 1

Vertex 1 ---> Color 2

Vertex 2 ---> Color 3

Vertex 3 ---> Color 1

**36.Write a program to compute container loader Problem.**

**C Code:**

**Here's a C program to solve the Container Loader Problem using the First-Fit Decreasing**

heuristic:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_ITEMS 100

#define MAX\_BINS 100

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

return (\*(int\*)b - \*(int\*)a); // Sort in descending order

}

void containerLoader(int items[], int n, int binCapacity) {

int bins[MAX\_BINS];

int binCount = 0;

int i, j;

for (i = 0; i < MAX\_BINS; i++) {

bins[i] = 0;

}

qsort(items, n, sizeof(int), compare);

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

int item = items[i];

int placed = 0;

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

if (bins[j] + item <= binCapacity) {

bins[j] += item;

placed = 1;

break;

}

}

if (!placed) {

bins[binCount] = item;

binCount++;

}

}

printf("Number of bins used: %d\n", binCount);

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

printf("Bin %d: %d\n", i + 1, bins[i]);

}

}

int main() {

int items[MAX\_ITEMS];

int n, binCapacity;

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

scanf("%d", &n);

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

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

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

}

printf("Enter the bin capacity: ");

scanf("%d", &binCapacity);

containerLoader(items, n, binCapacity);

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of items: 7

Enter the items:

5 3 8 6 2 7 4

Enter the bin capacity: 10

**Sample Output:**

Number of bins used: 4

Bin 1: 8 2

Bin 2: 7 3

Bin 3: 6 4

Bin 4: 5

**37.Write a program to generate the list of all factor for n value.**

**C Code**

#include <stdio.h>

void printFactors(int n) {

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

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

if (n % i == 0) { // If i is a factor of n

printf("%d ", i);

}

}

printf("\n");

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

printFactors(n);

return 0;

}

**Sample Input and Output**

**Sample Input:**

Enter a number: 36

**Sample Output:**

Factors of 36 are:

1 2 3 4 6 9 12 18 36

**38.Write a program to perform Assignment problem using branch and bound**

**C Code**

#include <stdio.h>

#include <limits.h>

#define N 4

void assignmentProblem(int costMatrix[N][N]);

int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,

int bound, int currCost, int minCost, int visited[]);

int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int

row, int visited[]);

int findMinCost(int costMatrix[N][N], int assignment[], int n, int

currCost, int minCost, int visited[]);

int main() {

int costMatrix[N][N] = {

{10, 2, 8, 12},

{9, 4, 7, 6},

{5, 11, 13, 10},

{7, 9, 16, 5}

};

assignmentProblem(costMatrix);

return 0;

}

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

int assignment[N] = {-1};

int visited[N] = {0}; // Track visited nodes

int minCost = INT\_MAX; minCost = branchAndBound(costMatrix, assignment, 0, N, 0, 0, minCost,

visited);

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

}

int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,

int bound, int currCost, int minCost, int visited[]) {

if (row == n) {

if (currCost < minCost) {

minCost = currCost;

}

return minCost;

}

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

if (!visited[col]) {

visited[col] = 1;

assignment[row] = col;

int newBound = bound + costMatrix[row][col];

int lowerBound = calculateLowerBound(costMatrix, assignment, n,

row + 1, visited);

far, explore further

if (newBound + lowerBound < minCost) {

minCost = branchAndBound(costMatrix, assignment, row + 1,

n, newBound, currCost + costMatrix[row][col], minCost, visited);

}

visited[col] = 0;

assignment[row] = -1;

}

}

return minCost;

}

int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int

row, int visited[]) {

int bound = 0;

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

int min1 = INT\_MAX, min2 = INT\_MAX;

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

if (!visited[j] && costMatrix[i][j] < min1) {

min2 = min1;

min1 = costMatrix[i][j];

} else if (!visited[j] && costMatrix[i][j] < min2) {

min2 = costMatrix[i][j];

}

}

bound += (min1 == INT\_MAX) ? 0 : min1;

bound += (min2 == INT\_MAX) ? 0 : min2;

}

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

int min1 = INT\_MAX, min2 = INT\_MAX;

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

if (!visited[j] && costMatrix[i][j] < min1) {

min2 = min1;

min1 = costMatrix[i][j];

} else if (!visited[j] && costMatrix[i][j] < min2) {

min2 = costMatrix[i][j];

}

}

bound += (min1 == INT\_MAX) ? 0 : min1;

bound += (min2 == INT\_MAX) ? 0 : min2;

}

return bound / 2;

}

**Sample Input and Output:**

**Sample Input:**

The cost matrix is hardcoded in the program:

10 2 8 12

9 4 7 6

5 11 13 10

7 9 16 5

**Sample Output:**

Minimum cost is 26

**39.Write a program for to perform liner search.**

**‘C’ code:**

#include <stdio.h>

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

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

if (arr[i] == target) {

return i; }

}

return -1; }

int main() {

int arr[100];

int size, target, result;

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

scanf("%d", &size);

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

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

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

}

printf("Enter the element to search for: ");

scanf("%d", &target);

result = linearSearch(arr, size, target);

if (result != -1) {

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

} else {

printf("Element %d not found in the array.\n", target);

}

return 0;

}

**Sample Input and Output:**

**Sample Input:**

Enter the number of elements in the array: 5

Enter the elements of the array:

10 20 30 40 50

Enter the element to search for: 30

**Sample Output:**

Element 30 found at index 2.

**40.Write a program to find out Hamiltonian circuit Using backtracking method**

**C Code**

**Here's a C program to find a Hamiltonian Circuit using the backtracking method:**

#include <stdio.h>

#include <stdbool.h>

#define V 5

bool isSafe(int graph[V][V], int path[], int pos) {

vertex.

if (graph[path[pos-1]][path[pos]] == 0) {

return false;

}

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

if (path[i] == path[pos]) {

return false;

}

}

return true;

}

bool hamCycleUtil(int graph[V][V], int path[], int pos) {

if (pos == V) {

first vertex

return graph[path[pos-1]][path[0]] == 1;

}

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

if (isSafe(graph, path, pos)) {

path[pos] = v;

if (hamCycleUtil(graph, path, pos + 1)) {

return true;

}

path[pos] = -1;

}

}

return false;

}

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

int path[V];

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

path[i] = -1;

}

path[0] = 0;

if (hamCycleUtil(graph, path, 1) == false) {

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

} else {

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

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

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

}

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

}

}

int main() {

int graph[V][V] = {

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

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

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

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

{0, 1, 1, 1, 0}

};

findHamiltonianCircuit(graph);

return 0;

}

**Sample Input and Output:**

**Sample Output:**

Hamiltonian Circuit found:

0 1 2 3 4 0