**Design and Analysis of Algorithm Lab**

1. **Fibonacci using recursion**

**Code:**

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1)

return n;

else

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;

}

**Output:**

Enter the number of terms: 10

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

--------------------------------

Process exited after 2.688 seconds with return value 0

Press any key to continue . . .

**Fibonacci even sum using recursion:**

#include <stdio.h>

int calculateEvenSum(int n)

{

if (n <= 0)

return 0;

int fibo[2 \* n + 1];

fibo[0] = 0, fibo[1] = 1;

int sum = 0;

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

{

fibo[i] = fibo[i - 1] + fibo[i - 2];

if (i % 2 == 0)

sum += fibo[i];

}

return sum;

}

int main()

{

int n;

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

scanf("%d", &n);

int sum = calculateEvenSum(n);

printf("Even indexed Fibonacci Sum upto %d terms = %d", n, sum);

return 0;

}

**OUTPUT:**

Enter the value of n: 5

Even indexed Fibonacci Sum upto 5 terms = 88

--------------------------------

Process exited after 1.412 seconds with return value 0

Press any key to continue . . .

1. **Armstrong number with recursion**

**Code:**

#include <stdio.h>

#include <math.h>

int countDigits(int num) {

if (num == 0)

return 0;

else

return 1 + countDigits(num / 10);

}

int isArmstrong(int num, int sum, int n) {

if (num == 0)

return sum;

else

return isArmstrong(num / 10, sum + pow(num % 10, n), n);

}

int main() {

int num, sum = 0, digitCount, result;

printf("Enter a number: ");

scanf("%d", &num);

digitCount = countDigits(num);

result = isArmstrong(num, sum, digitCount);

if (result == num)

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

else

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

return 0;

}

**Output:**

Enter a number: 56

56 is not an Armstrong number.

--------------------------------

Process exited after 4.186 seconds with return value 0

Press any key to continue . . .

1. **GCD using two numbers**

**Code:**

#include <stdio.h>

int min(int a, int b) {

return (a < b) ? a : b;

}

int gcd(int a, int b, int i) {

if (i == 1)

return 1;

else if (a % i == 0 && b % i == 0)

return i;

else

return gcd(a, b, i - 1);

}

int main() {

int num1, num2, result;

printf("Enter two numbers: ");

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

result = gcd(num1, num2, min(num1, num2));

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

return 0;

}

**Output:**

Enter two numbers: 56 23

GCD of 56 and 23 is 1

--------------------------------

Process exited after 165.9 seconds with return value 0

Press any key to continue . . .

1. **Largest element of array**

**Code:**

#include <stdio.h>

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

int i;

int largest = arr[0];

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

if (arr[i] > largest) {

largest = arr[i];

}

}

return largest;

}

int main() {

int arr[100], n, i;

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

scanf("%d", &n);

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

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

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

}

int largest = findLargest(arr, n);

printf("The largest element in the array is: %d\n", largest);

return 0;

}

**Output:**

Enter the number of elements in the array: 10

Enter 10 elements:

1

2

3

9

8

7

6

4

5

100

The largest element in the array is: 100

--------------------------------

Process exited after 18.94 seconds with return value 0

Press any key to continue . . .

1. **Factorial using recursion**

**Code:**

#include <stdio.h>

unsigned long long factorial(int n) {

if (n == 0 || n == 1)

return 1;

else

return n \* factorial(n - 1);

}

int main() {

int num;

unsigned long long fact;

printf("Enter a non-negative integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Factorial is not defined for negative numbers.\n");

return 1;

}

fact = factorial(num);

printf("Factorial of %d = %llu\n", num, fact);

return 0;

}

**Output:**

Enter a non-negative integer: 6

Factorial of 6 = 720

--------------------------------

Process exited after 1.102 seconds with return value 0

Press any key to continue . . .

1. **Prime checker using recursion**

**Code:**

#include <stdio.h>

#include <stdbool.h>

bool isPrime(int num, int i) {

if (num <= 2) {

if (num == 2)

return true;

else

return false;

}

if (num % i == 0)

return false;

if (i \* i > num)

return true;

return isPrime(num, i + 1);

}

int main() {

int num;

printf("Enter a positive integer: ");

scanf("%d", &num);

if (num <= 1) {

printf("The number is not prime.\n");

} else {

if (isPrime(num, 2))

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

else

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

}

return 0;

}

**Output:**

Enter a positive integer: 19

19 is a prime number.

--------------------------------

Process exited after 18.85 seconds with return value 0

Press any key to continue . . .

1. **Selection sort**

**Code:**

#include <stdio.h>

void selectionSort(int arr[], 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 (arr[j] < arr[min\_index])

min\_index = j;

}

temp = arr[min\_index];

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

arr[i] = temp;

}

}

int main() {

int arr[100], n, i;

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

scanf("%d", &n);

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

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

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

}

selectionSort(arr, n);

printf("Array after sorting: ");

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

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

}

printf("\n");

return 0;

}

**Output:**

Enter the number of elements in the array: 10

Enter 10 elements:

8 3 7 4 1 7 0 7 4 3

Array after sorting: 0 1 3 3 4 4 7 7 7 8

--------------------------------

Process exited after 15.23 seconds with return value 0

Press any key to continue . . .

1. **Bubble sort**

**Code:**

#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 arr[100], n, i;

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

scanf("%d", &n);

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

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

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

}

bubbleSort(arr, n);

printf("Array after sorting: ");

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

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

}

printf("\n");

return 0;

}

**Output:**

Enter the number of elements in the array: 10

Enter 10 elements:

6 3 0 5 9 6 6 7 4 2

Array after sorting: 0 2 3 4 5 6 6 6 7 9

--------------------------------

Process exited after 9.437 seconds with return value 0

Press any key to continue . . .

1. **Time complexity of multiplying two matrices**

**Code:**

#include <stdio.h>

#define N 3

void multiplyMatrix(int mat1[][N], int mat2[][N], int result[][N]) {

int i, j, k;

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

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

result[i][j] = 0;

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

result[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

}

void displayMatrix(int mat[][N]) {

int i, j;

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

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

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

}

printf("\n");

}

}

int main() {

int mat1[N][N], mat2[N][N], result[N][N];

int i, j;

printf("Enter elements of matrix 1 (%d x %d):\n", N, N);

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

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

scanf("%d", &mat1[i][j]);

}

}

printf("Enter elements of matrix 2 (%d x %d):\n", N, N);

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

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

scanf("%d", &mat2[i][j]);

}

}

multiplyMatrix(mat1, mat2, result);

printf("\nMatrix 1:\n");

displayMatrix(mat1);

printf("\nMatrix 2:\n");

displayMatrix(mat2);

printf("\nResultant Matrix:\n");

displayMatrix(result);

return 0;

}

**Output:**

Enter elements of matrix 1 (3 x 3):

1 2 3 4 5 6 7 8 9

Enter elements of matrix 2 (3 x 3):

9 8 7 6 5 4 3 2 1

Matrix 1:

1 2 3

4 5 6

7 8 9

Matrix 2:

9 8 7

6 5 4

3 2 1

Resultant Matrix:

30 24 18

84 69 54

138 114 90

--------------------------------

Process exited after 11.57 seconds with return value 0

Press any key to continue . . .

1. **String palindrome or not using recursion**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#include <ctype.h>

bool isPalindrome(char str[], int start, int end) {

if (start >= end)

return true;

if (tolower(str[start]) != tolower(str[end]))

return false;

return isPalindrome(str, start + 1, end - 1);

}

int main() {

char str[100];

printf("Enter a string: ");

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

if (str[strlen(str) - 1] == '\n')

str[strlen(str) - 1] = '\0';

if (isPalindrome(str, 0, strlen(str) - 1))

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

else

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

return 0;

}

**Output:**

Enter a string: Malayalam

The string "Malayalam" is a palindrome.

--------------------------------

Process exited after 4.723 seconds with return value 0

Press any key to continue . . .

1. **Copy one string to another using recursion**

**Code:**

#include <stdio.h>

void stringCopy(char \*source, char \*destination) {

if (\*source == '\0') {

\*destination = '\0';

return;

}

\*destination = \*source;

stringCopy(source + 1, destination + 1);

}

int main() {

char source[100], destination[100];

printf("Enter a string: ");

scanf("%s", source);

stringCopy(source, destination);

printf("Original string: %s\n", source);

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

return 0;

}

**Output:**

Enter a string: SAVEETHA

Original string: SAVEETHA

Copied string: SAVEETHA

--------------------------------

Process exited after 45.57 seconds with return value 0

Press any key to continue . . .

1. **Binary Search**

**Code:**

#include <stdio.h>

// Function to perform binary search

int binarySearch(int arr[], int left, int right, int key) {

while (left <= right) {

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

// Check if the key is present at the middle

if (arr[mid] == key)

return mid;

// If key is greater, ignore left half

if (arr[mid] < key)

left = mid + 1;

// If key is smaller, ignore right half

else

right = mid - 1;

}

// Key not found

return -1;

}

int main() {

int n, key;

printf("Enter the size of the array: ");

scanf("%d", &n);

int arr[n];

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

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

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

}

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

scanf("%d", &key);

int result = binarySearch(arr, 0, n - 1, key);

if (result == -1)

printf("Element %d is not present in the array.\n", key);

else

printf("Element %d is present at index %d.\n", key, result);

return 0;

}

**Output:**

Enter the size of the array: 20

Enter the elements of the sorted array:

5 6 11 17 28 45 73 118 191 309 500 809 1309 2118 3427 5545 8927 14517 23444 37961

Enter the key to search: 2118

Element 2118 is present at index 13.

--------------------------------

Process exited after 275.9 seconds with return value 0

Press any key to continue . . .

1. **Reverse a string using recursion**

**Code:**

#include <stdio.h>

#include <string.h>

// Function to print the reverse of a string using recursion

void reverseString(char \*str) {

if (\*str == '\0') {

return;

}

reverseString(str + 1); // Recursively call for next character

printf("%c", \*str); // Print current character

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

printf("Reverse of the string: ");

reverseString(str);

printf("\n");

return 0;

}

**Output:**

Enter a string: Marvel

Reverse of the string: levraM

--------------------------------

Process exited after 30.82 seconds with return value 0

Press any key to continue . . .

1. **Maximum and minimum value sequence for all numbers in a list**

**Code:**

#include <stdio.h>

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

int min\_so\_far = arr[0];

int max\_so\_far = arr[0];

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

if (arr[i] < min\_so\_far)

min\_so\_far = arr[i];

if (arr[i] > max\_so\_far)

max\_so\_far = arr[i];

printf("Minimum sequence value for %d: %d\n", arr[i], min\_so\_far);

printf("Maximum sequence value for %d: %d\n", arr[i], max\_so\_far);

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

printf("\nMinimum and maximum value sequence for each number in the list:\n");

minMaxSequence(arr, n);

return 0;

}

**Output:**

Enter the number of elements in the list: 9

Enter the elements of the list:

8

4

8

4

6

0

9

9

6

Minimum and maximum value sequence for each number in the list:

Minimum sequence value for 8: 8

Maximum sequence value for 8: 8

Minimum sequence value for 4: 4

Maximum sequence value for 4: 8

Minimum sequence value for 8: 4

Maximum sequence value for 8: 8

Minimum sequence value for 4: 4

Maximum sequence value for 4: 8

Minimum sequence value for 6: 4

Maximum sequence value for 6: 8

Minimum sequence value for 0: 0

Maximum sequence value for 0: 8

Minimum sequence value for 9: 0

Maximum sequence value for 9: 9

Minimum sequence value for 9: 0

Maximum sequence value for 9: 9

Minimum sequence value for 6: 0

Maximum sequence value for 6: 9

--------------------------------

Process exited after 13.16 seconds with return value 0

Press any key to continue . . .

1. **Strassen’s matrix multiplication**

**Code:**

#include <stdio.h>

#define N 2

void strassen(int A[][N], int B[][N], int C[][N]);

int main() {

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

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

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

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

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

}

}

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

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

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

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

}

}

strassen(A, B, C);

printf("Result Matrix:\n");

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

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

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

}

printf("\n");

}

return 0;

}

void strassen(int A[][N], int B[][N], int C[][N]) {

int M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

int M2 = (A[1][0] + A[1][1]) \* B[0][0];

int M3 = A[0][0] \* (B[0][1] - B[1][1]);

int M4 = A[1][1] \* (B[1][0] - B[0][0]);

int M5 = (A[0][0] + A[0][1]) \* B[1][1];

int M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

int M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

C[0][0] = M1 + M4 - M5 + M7;

C[0][1] = M3 + M5;

C[1][0] = M2 + M4;

C[1][1] = M1 - M2 + M3 + M6;

}

**Output:**

Enter elements of matrix A (2x2):

5 6 7 8

Enter elements of matrix B (2x2):

1 2 3 4

Result Matrix:

23 34

31 46

--------------------------------

Process exited after 12.03 seconds with return value 0

Press any key to continue . . .

1. **Merge sort**

**Code:**

#include <stdio.h>

// Merge function to merge two subarrays

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

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

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

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

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

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

// Merge the temporary arrays back into arr[left..right]

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++;

}

// Copy the remaining elements of L[], if any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of R[], if any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort function

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

if (left < right) {

// Find the middle point

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

// Sort first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

// Function to print an array

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

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

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

printf("\n");

}

// Main function

int main() {

int arr[100], n;

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("Given array is \n");

printArray(arr, n);

mergeSort(arr, 0, n - 1);

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

printArray(arr, n);

return 0;

}

**Output:**

Enter the number of elements: 5

Enter the elements:

55

11

99

77

33

Given array is

55 11 99 77 33

Sorted array is

11 33 55 77 99

--------------------------------

Process exited after 11.78 seconds with return value 0

Press any key to continue . . .

1. **Divide and conquer to find maximum and minimum value in list**

**Code:**

#include <stdio.h>

// Structure to store the result of max and min values

struct Pair {

int min;

int max;

};

// Function to find the maximum and minimum values in a list using Divide and Conquer

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

struct Pair result, left, right;

int mid;

// If there is only one element

if (low == high) {

result.min = arr[low];

result.max = arr[low];

return result;

}

// If there are two elements

if (high == low + 1) {

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

result.max = arr[low];

result.min = arr[high];

} else {

result.max = arr[high];

result.min = arr[low];

}

return result;

}

// If there are more than two elements, divide the array into two halves

mid = (low + high) / 2;

left = findMaxMin(arr, low, mid);

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

// Compare minimums of two halves

if (left.min < right.min)

result.min = left.min;

else

result.min = right.min;

// Compare maximums of two halves

if (left.max > right.max)

result.max = left.max;

else

result.max = right.max;

return result;

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

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

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

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

return 0;

}

**Output:**

Enter the number of elements in the list:

6

Enter the elements of the list:

1

4

6

1

032

2

Minimum element: 1

Maximum element: 32

--------------------------------

Process exited after 21.62 seconds with return value 0

Press any key to continue . . .

1. **Generate all prime numbers using recursion**

**Code:**

#include <stdio.h>

#include <stdbool.h>

bool isPrime(int num, int i) {

if (i == 1) // Base case

return true;

if (num % i == 0)

return false;

return isPrime(num, i - 1); // Recursively check for divisibility

}

void generatePrimes(int n, int i) {

if (i > n)

return;

if (isPrime(i, i / 2)) // Check if i is prime

printf("%d ", i);

generatePrimes(n, i + 1); // Recursively generate primes up to n

}

int main() {

int n;

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

scanf("%d", &n);

printf("Prime numbers up to %d are: ", n);

generatePrimes(n, 2); // Start generating primes from 2

printf("\n");

return 0;

}

**Output:**

Enter the value of n: 10

Prime numbers up to 10 are: 2 3 5 7

--------------------------------

Process exited after 3.074 seconds with return value 0

Press any key to continue . . .

1. **Knapsack using greedy techniques**

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent items

struct Item {

int weight;

int value;

};

// Function to compare items based on their value/weight ratio

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

double ratio1 = (double)(((struct Item\*)a)->value / (double)((struct Item\*)a)->weight);

double ratio2 = (double)(((struct Item\*)b)->value / (double)((struct Item\*)b)->weight);

if (ratio1 < ratio2)

return 1;

else if (ratio1 > ratio2)

return -1;

else

return 0;

}

// Function to solve fractional knapsack problem

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

// Sort items based on value/weight ratio

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

double totalValue = 0.0; // Total value of items in knapsack

int currentWeight = 0; // Current weight in knapsack

// Iterate through sorted items and add to knapsack as much as possible

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

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

// If the whole item can be added

currentWeight += items[i].weight;

totalValue += items[i].value;

} else {

// Otherwise, add a fraction of the item

int remainingWeight = capacity - currentWeight;

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

break; // No more items can be added

}

}

return totalValue;

}

int main() {

int capacity; // Capacity of knapsack

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

scanf("%d", &capacity);

int n; // Number of items

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

scanf("%d", &n);

struct Item items[n]; // Array of items

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

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

printf("Item %d:\n", i + 1);

printf("Weight: ");

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

printf("Value: ");

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

}

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

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

return 0;

}

**Output:**

Enter the capacity of the knapsack: 30

Enter the number of items: 4

Enter the weight and value of each item:

Item 1:

Weight: 3

Value: 40

Item 2:

Weight: 15

Value: 20

Item 3:

Weight: 10

Value: 10

Item 4:

Weight: 15

Value: 40

Maximum value in Knapsack = 96.00

--------------------------------

Process exited after 53.62 seconds with return value 0

Press any key to continue . . .

1. **MST using greedy techniques**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

#define V 100 // Maximum number of vertices

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

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int graph[V][V], int n) {

printf("Edge \tWeight\n");

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

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

}

void primMST(int graph[V][V], int n) {

int parent[V]; // Array to store constructed MST

int key[V]; // Key values used to pick the minimum weight edge in cut

bool mstSet[V]; // To represent set of vertices not yet included in MST

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0; // Make key 0 so that this vertex is picked as first vertex

parent[0] = -1; // First node is always root of MST

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

int u = minKey(key, mstSet);

mstSet[u] = true;

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

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

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

}

printMST(parent, graph, n);

}

int main() {

int n; // Number of vertices

printf("Enter the number of vertices (maximum %d): ", V);

scanf("%d", &n);

int graph[V][V]; // Adjacency matrix representing the graph

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]);

}

}

// Print the MST

printf("Minimum Spanning Tree:\n");

primMST(graph, n);

return 0;

}

**Output:**

Enter the number of vertices (maximum 100): 5

Enter the adjacency matrix:

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

Minimum Spanning Tree:

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

--------------------------------

Process exited after 34.97 seconds with return value 0

Press any key to continue . . .

1. **Knapsack using dynamic programming**

**Code:**

#include<stdio.h>

// Function to find maximum of two integers

int max(int a, int b) { return (a > b)? a : b; }

// Function to solve knapsack problem using dynamic programming

int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

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

// Build table K[][] in bottom up manner

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

{

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

{

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

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

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

}

}

return K[n][W];

}

int main()

{

int n, W;

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

scanf("%d", &n);

int val[n], wt[n];

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

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

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

}

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

scanf("%d", &W);

printf("The maximum value that can be put in a knapsack of capacity %d is: %d", W, knapSack(W, wt, val, n));

return 0;

}

**Output:**

Enter the number of items: 4

Enter the values and weights of the items:

40

15

12

56

45

23

12

47

Enter the capacity of the knapsack: 150

The maximum value that can be put in a knapsack of capacity 150 is: 109

--------------------------------

Process exited after 22.73 seconds with return value 0

Press any key to continue . . .

1. **Optimal binary search using dynamic programming**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// A utility function to get sum of array elements freq[i] to freq[j]

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

{

int s = 0;

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

s += freq[k];

return s;

}

// A utility function to print the constructed BST

void printOptimalBST(int root[], int i, int j, int level, int n)

{

if (i > j) {

printf("Level %d: Null\n", level);

return;

}

int mid = root[i \* n + j];

printf("Level %d: %d\n", level, mid);

printOptimalBST(root, i, mid - 1, level + 1, n);

printOptimalBST(root, mid + 1, j, level + 1, n);

}

// Function to construct optimal BST and return its cost

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

{

// Create an auxiliary 2D matrix to store results of subproblems

int cost[n][n];

// root[i][j] will store the index of the root of the optimal BST that includes keys[i] to keys[j]

int root[n \* n];

// Initialize cost and root arrays as INT\_MAX

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

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

root[i \* n + i] = i;

}

// Build the table cost[][] in a bottom-up manner

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

for (int i = 0; i <= n - L; 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;

root[i \* n + j] = r;

}

}

}

}

printf("Optimal BST structure:\n");

printOptimalBST(root, 0, n - 1, 0, n);

return cost[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:\n");

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

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

}

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

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

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

}

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

return 0;

}

**Output:**

Enter the number of keys: 5

Enter the keys:

4

8

6

7

3

Enter the frequencies:

1

2

3

4

5

Optimal BST structure:

Level 0: 3

Level 1: 1

Level 2: 0

Level 3: Null

Level 3: Null

Level 2: 2

Level 3: Null

Level 3: Null

Level 1: 4

Level 2: Null

Level 2: Null

Cost of optimal BST is 30

--------------------------------

Process exited after 13.65 seconds with return value 0

Press any key to continue . . .

1. **Binomial coefficient using dynamic programming**

**Code:**

#include <stdio.h>

// Function to calculate binomial coefficient C(n, k)

unsigned long long binomialCoeff(int n, int k)

{

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

int i, j;

// Calculate value of binomial coefficient in bottom-up manner

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

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

// Base Cases

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

C[i][j] = 1;

// Calculate value using previously stored values

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);

if (k > n) {

printf("Invalid input: k must be less than or equal to n.\n");

return 1;

}

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

return 0;

}

**Output:**

Enter the value of n: 15

Enter the value of k: 3

Binomial coefficient C(15, 3) is 455

--------------------------------

Process exited after 4.905 seconds with return value 0

Press any key to continue . . .

1. **Reverse a number using recursive**

**Code:**

#include <stdio.h>

// Recursive function to find the reverse of a number

int reverse(int num, int rev) {

// Base case: when num becomes 0

if (num == 0)

return rev;

// Extract the last digit of num and append it to rev

return reverse(num / 10, rev \* 10 + num % 10);

}

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Call the recursive function to find the reverse

int reversedNum = reverse(num, 0);

// Output the reversed number

printf("Reverse of %d is %d\n", num, reversedNum);

return 0;

}

**Output:**

Enter a number: 456

Reverse of 456 is 654

--------------------------------

Process exited after 1.91 seconds with return value 0

Press any key to continue . . .

1. **Perfect number**

**Code:**

#include <stdio.h>

// Function to check if a number is perfect or not

int isPerfect(int num) {

int sum = 0;

// Calculate the sum of proper divisors

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

if (num % i == 0) {

sum += i;

}

}

// Check if the sum equals the original number

return sum == num;

}

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Check if the number is perfect and print the result

if (isPerfect(num)) {

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

} else {

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

}

return 0;

}

**Output:**

Enter a number: 6

6 is a perfect number.

--------------------------------

Process exited after 2.414 seconds with return value 0

Press any key to continue . . .

1. **Pattern**

**Code:**

#include <stdio.h>

// Recursive function to print the pattern

void printPattern(int n, int row, int col) {

// Base case: if the current row exceeds n, return

if (row > n)

return;

// Print numbers from 1 to col

if (col <= row) {

printf("%d ", col);

printPattern(n, row, col + 1);

} else {

// Move to the next row

printf("\n");

printPattern(n, row + 1, 1);

}

}

int main() {

int n;

// Input value of n from the user

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

scanf("%d", &n);

// Call the recursive function to print the pattern

printPattern(n, 1, 1);

return 0;

}

**Output:**

Enter the value of n: 4

1

1 2

1 2 3

1 2 3 4

--------------------------------

Process exited after 1.949 seconds with return value 0

Press any key to continue . . .

1. **Travelling salesman using dynamic programming**

**Code:**

#include <stdio.h>

#include <limits.h>

#define V 10

int min(int x, int y) { return (x < y) ? x : y; }

int tsp(int graph[][V], int mask, int pos, int n, int dp[][V]) {

if (mask == (1 << n) - 1) return graph[pos][0];

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

int ans = INT\_MAX;

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

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

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

ans = min(ans, newAns);

}

return dp[mask][pos] = ans;

}

int main() {

int n;

printf("Enter the number of vertices (maximum %d): ", V);

scanf("%d", &n);

int graph[V][V], dp[1 << V][V];

printf("Enter the cost matrix (%d x %d):\n", n, n);

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

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

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

for (int j = 0; j < V; j++) dp[i][j] = -1;

int minCost = tsp(graph, 1, 0, n, dp);

printf("Minimum cost of the Hamiltonian cycle: %d\n", minCost);

return 0;

}

**Output:**

Enter the number of vertices (maximum 10): 4

Enter the cost matrix (4 x 4):

0 10 15 20

10 0 35 25

15 35 0 30

20 25 30 0

Minimum cost of the Hamiltonian cycle: 80

--------------------------------

Process exited after 23.85 seconds with return value 0

Press any key to continue . . .

1. **Floyd’s algorithm**

**Code:**

#include <stdio.h>

#include <limits.h>

#define V 10 // Maximum number of vertices in the graph

// Function to print the distance matrix

void printSolution(int dist[][V], int n) {

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] == INT\_MAX)

printf("INF\t");

else

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

}

printf("\n");

}

}

// Function to perform Floyd's algorithm

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

int dist[V][V]; // Output matrix that will have the shortest distances between every pair of vertices

// Initialize distance matrix

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

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

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

// Update distance matrix by considering all vertices as intermediate vertices

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

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

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

// If vertex k is on the shortest path from i to j, then update the value of dist[i][j]

if (dist[i][k] != INT\_MAX && dist[k][j] != INT\_MAX && dist[i][k] + dist[k][j] < dist[i][j])

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

}

}

}

// Print the shortest distances

printSolution(dist, n);

}

int main() {

int n;

// Input the number of vertices in the graph

printf("Enter the number of vertices in the graph (maximum %d): ", V);

scanf("%d", &n);

// Input the adjacency matrix representing the graph

int graph[V][V];

printf("Enter the adjacency matrix representing the graph (INF for no edge):\n");

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

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

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

}

}

// Run Floyd's algorithm

floydWarshall(graph, n);

return 0;

}

**Output:**

Enter the number of vertices in the graph (maximum 10): 5

Enter the adjacency matrix representing the graph (INF for no edge):

1 2 3 4 5

0 1 2 3 4

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

Shortest distances between every pair of vertices:

1 2 3 4 5

0 1 2 3 4

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

--------------------------------

Process exited after 27.09 seconds with return value 0

Press any key to continue . . .

1. **Pascal triangle**

**Code:**

#include <stdio.h>

// Function to calculate the factorial of a number

int factorial(int n) {

if (n == 0 || n == 1)

return 1;

else

return n \* factorial(n - 1);

}

// Function to calculate the binomial coefficient C(n, k)

int binomialCoeff(int n, int k) {

return factorial(n) / (factorial(k) \* factorial(n - k));

}

// Function to print Pascal's triangle

void printPascalTriangle(int numRows) {

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

// Print spaces to align the triangle

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

printf(" ");

}

// Print values for the current row

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

printf("%d ", binomialCoeff(i, j));

}

printf("\n");

}

}

int main() {

int numRows;

// Input the number of rows for Pascal's triangle from the user

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

scanf("%d", &numRows);

// Print Pascal's triangle

printPascalTriangle(numRows);

return 0;

}

**Output:**

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

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

--------------------------------

Process exited after 2.275 seconds with return value 0

Press any key to continue . . .

1. **Sum of digits**

**Code:**

#include <stdio.h>

#include <stdlib.h> // For abs function

int main() {

int num, sum = 0, digit, i = 0;

int digits[10]; // Array to store individual digits

// Input number from user

printf("Enter a number: ");

scanf("%d", &num);

// Take the absolute value of the number

num = abs(num);

// Calculate sum of digits and store them in the array

while (num != 0) {

digit = num % 10; // Get the last digit

sum += digit; // Add the digit to sum

digits[i++] = digit; // Store the digit in the array

num /= 10; // Remove the last digit from the number

}

// Display the sum of digits in the desired format

printf("Sum of digits = ");

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

printf("%d", digits[j]); // Print the digit

if (j != 0) {

printf("+");

} else {

printf("=");

}

}

printf("%d\n", sum); // Print the sum

return 0;

}

**Output:**

Enter a number: -856

Sum of digits = 8+5+6=19

--------------------------------

Process exited after 6.003 seconds with return value 0

Press any key to continue . . .

1. **Linear search**

**Code:**

#include <stdio.h>

// Function to perform linear search

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

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

if (arr[i] == key) {

return i; // Return the index where the key is found

}

}

return -1; // Return -1 if the key is not found

}

int main() {

int arr[100], n, key, index;

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

scanf("%d", &n);

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

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

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

}

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

scanf("%d", &key);

index = linearSearch(arr, n, key);

if (index != -1) {

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

} else {

printf("Element not found\n");

}

return 0;

}

**Output:**

Enter the number of elements in the array: 10

Enter the elements of the array: 1 5 9 7 3 8 2 4 6 0

Enter the element to search: 3

Element found at index 4

--------------------------------

Process exited after 26.73 seconds with return value 0

Press any key to continue . . .

1. **N-Queens Problem**

**Code:**

#include <stdio.h>

#include <stdbool.h>

// Function to print the solution

void printSolution(int board[][10], int N) {

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

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

printf("%2d ", board[i][j]);

}

printf("\n");

}

}

// Function to check if a queen can be placed on the board at position (row, col)

bool isSafe(int board[][10], int row, int col, int N) {

int i, j;

// Check this row on the left side

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

if (board[row][i])

return false;

}

// Check upper diagonal on the left side

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

if (board[i][j])

return false;

}

// Check lower diagonal on the left side

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

if (board[i][j])

return false;

}

return true;

}

// Recursive function to solve N-Queens problem using backtracking

bool solveNQueensUtil(int board[][10], int col, int N) {

// If all queens are placed, return true

if (col >= N)

return true;

// Consider this column and try placing this queen in all rows one by one

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

// Check if the queen can be placed on the board[i][col]

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

// Place this queen in board[i][col]

board[i][col] = 1;

// Recur to place rest of the queens

if (solveNQueensUtil(board, col + 1, N))

return true;

// If placing queen in board[i][col] doesn't lead to a solution, then remove queen from board[i][col]

board[i][col] = 0; // BACKTRACK

}

}

// If the queen cannot be placed in any row in this column, then return false

return false;

}

// Function to solve the N-Queens problem

bool solveNQueens(int N) {

int board[10][10] = {0};

if (solveNQueensUtil(board, 0, N) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board, N);

return true;

}

// Main function

int main() {

int N;

printf("Enter the size of the board: ");

scanf("%d", &N);

if (N <= 0 || N > 10) {

printf("Invalid input. Size should be between 1 and 10.\n");

return 1;

}

solveNQueens(N);

return 0;

}

**Output:**

Enter the size of the board: 8

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0

--------------------------------

Process exited after 1.396 seconds with return value 0

Press any key to continue . . .

1. **Sum of subsets using backtracking**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 100

// Function to print the subset

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

printf("{ ");

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

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

}

printf("}\n");

}

// Function to find all subsets with the given sum using backtracking

void findSubsets(int set[], int subset[], int n, int subsetSize, int sum, int targetSum, int k) {

if (sum == targetSum) {

printSubset(subset, subsetSize);

findSubsets(set, subset, n, subsetSize - 1, sum - set[k], targetSum, k + 1);

return;

} else {

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

subset[subsetSize] = set[i];

findSubsets(set, subset, n, subsetSize + 1, sum + set[i], targetSum, i + 1);

}

}

}

// Function to initialize the variables and call findSubsets function

void sumOfSubsets(int set[], int n, int targetSum) {

int subset[MAX\_SIZE] = {0};

findSubsets(set, subset, n, 0, 0, targetSum, 0);

}

// Main function

int main() {

int set[MAX\_SIZE], n, targetSum;

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

scanf("%d", &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", &targetSum);

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

sumOfSubsets(set, n, targetSum);

return 0;

}

**Output:**

Enter the number of elements in the set: 7

Enter the elements of the set: 7 14 21 28 35 42 49

Enter the target sum: 56

Subsets with sum 56 are:

{ 7 14 35 }

{ 7 21 28 }

{ 7 49 }

{ 14 42 }

{ 21 35 }

--------------------------------

Process exited after 51.21 seconds with return value 0

Press any key to continue . . .

1. **Graph colouring using backtracking**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

// Function to print the solution

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

printf("Vertex\tColor\n");

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

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

}

}

// Function to check if the current color assignment is safe for the vertex v

bool isSafe(int v, bool graph[][MAX\_VERTICES], int color[], int c, int V) {

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

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

return false;

}

}

return true;

}

// Recursive function to solve the graph coloring problem using backtracking

bool graphColoringUtil(bool graph[][MAX\_VERTICES], int m, int color[], int v, int V) {

if (v == V) {

return true; // All vertices are assigned a color

}

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

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

color[v] = c;

// Recur to assign colors to rest of the vertices

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

return true;

}

// If assigning color c doesn't lead to a solution, then remove it

color[v] = 0;

}

}

return false; // No color can be assigned to this vertex

}

// Function to solve the graph coloring problem

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

int color[MAX\_VERTICES] = {0}; // Initialize all vertices as uncolored

if (!graphColoringUtil(graph, m, color, 0, V)) {

printf("Solution does not exist\n");

return false;

}

printf("Solution exists with following color assignments:\n");

printSolution(color, V);

return true;

}

// Main function

int main() {

int V, m;

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

scanf("%d", &V);

printf("Enter the adjacency matrix (0/1):\n");

bool graph[MAX\_VERTICES][MAX\_VERTICES];

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

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

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

}

}

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

scanf("%d", &m);

graphColoring(graph, m, V);

return 0;

}

**Output:**

Enter the number of vertices in the graph: 4

Enter the adjacency matrix (0/1):

0 1 1 1

1 0 1 0

1 1 0 1

1 0 1 0

Enter the number of colors: 4

Solution exists with following color assignments:

Vertex Color

0 1

1 2

2 3

3 2

--------------------------------

Process exited after 27.81 seconds with return value 0

Press any key to continue . . .

1. **Hamiltonian circuit**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

// Function to check if the vertex v can be added at index 'pos' in the Hamiltonian cycle

bool isSafe(int v, bool graph[][MAX\_VERTICES], int path[], int pos, int V) {

// Check if this vertex is adjacent to the previously added vertex and not already added

if (!graph[path[pos - 1]][v])

return false;

// Check if the vertex has already been included

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

if (path[i] == v)

return false;

return true;

}

// Recursive utility function to find a Hamiltonian circuit starting from the vertex 'pos'

bool hamiltonianCircuitUtil(bool graph[][MAX\_VERTICES], int path[], int pos, int V) {

// If all vertices are included in the Hamiltonian cycle

if (pos == V) {

// Check if there is an edge from the last vertex to the first vertex

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

return true;

else

return false;

}

// Try different vertices as the next candidate in Hamiltonian cycle

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

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

path[pos] = v;

// Recur to construct the rest of the path

if (hamiltonianCircuitUtil(graph, path, pos + 1, V))

return true;

// If adding vertex v doesn't lead to a solution, remove it

path[pos] = -1;

}

}

return false;

}

// Function to find and print the Hamiltonian circuit in the given graph

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

int path[MAX\_VERTICES];

// Initialize all vertices as not included in the Hamiltonian cycle

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

path[i] = -1;

// Start from vertex 0 as the first vertex in the path

path[0] = 0;

if (!hamiltonianCircuitUtil(graph, path, 1, V)) {

printf("Hamiltonian circuit does not exist\n");

return;

}

printf("Hamiltonian circuit exists: ");

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

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

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

}

// Main function

int main() {

int V;

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

scanf("%d", &V);

printf("Enter the adjacency matrix (0/1) for the graph:\n");

bool graph[MAX\_VERTICES][MAX\_VERTICES];

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

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

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

}

}

hamiltonianCircuit(graph, V);

return 0;

}

**Output:**

Enter the number of vertices in the graph: 4

Enter the adjacency matrix (0/1) for the graph:

0 1 1 0

1 0 1 1

1 1 0 1

0 1 1 0

Hamiltonian circuit exists: 0 1 3 2 0

--------------------------------

Process exited after 42 seconds with return value 0

Press any key to continue . . .

1. **Container loading**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_ITEMS 100

#define MAX\_CONTAINERS 100

// Function to load items into containers

int loadContainers(int items[], int n, int capacity) {

int containers[MAX\_CONTAINERS] = {0};

int containerCount = 0;

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

int j;

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

if (containers[j] >= items[i]) {

containers[j] -= items[i];

break;

}

}

if (j == containerCount) {

containerCount++;

containers[j] = capacity - items[i];

}

}

return containerCount;

}

int main() {

int items[MAX\_ITEMS], n, capacity;

// Input the number of items

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

scanf("%d", &n);

// Input the dimensions of each item

printf("Enter the dimensions of each item:\n");

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

printf("Item %d: ", i + 1);

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

}

// Input the capacity of each container

printf("Enter the capacity of each container: ");

scanf("%d", &capacity);

// Compute the number of containers needed

int containersNeeded = loadContainers(items, n, capacity);

printf("Minimum number of containers needed: %d\n", containersNeeded);

return 0;

}

**Output:**

Enter the number of items: 5

Enter the dimensions of each item:

Item 1: 5

Item 2: 2

Item 3: 8

Item 4: 9

Item 5: 1

Enter the capacity of each container: 16

Minimum number of containers needed: 2

--------------------------------

Process exited after 12.57 seconds with return value 0

Press any key to continue . . .

1. **Factors of n numbers**

**Code:**

#include <stdio.h>

void printFactors(int n, int i) {

if (i > n)

return;

if (n % i == 0) {

printf("%d ", i);

printFactors(n, i + 1);

} else {

printFactors(n, i + 1);

}

}

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d", &n);

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

printFactors(n, 1);

return 0;

}

**Output:**

Enter a positive integer: 36

Factors of 36 are: 1 2 3 4 6 9 12 18 36

--------------------------------

Process exited after 33.21 seconds with return value 0

Press any key to continue . . .

1. **Job assignment problem**

**Code:**

#include <stdio.h>

#include <limits.h>

#define N 4 // Number of workers and tasks, assuming square matrix for simplicity

// Function to find the minimum cost assignment using Branch and Bound

void branchAndBound(int costMatrix[N][N], int assigned[N], int worker, int cost, int \*minCost) {

// Base case: if all workers are assigned tasks, update minimum cost

if (worker == N) {

if (cost < \*minCost)

\*minCost = cost;

return;

}

// Prune the search if the current cost exceeds the minimum cost

if (cost >= \*minCost)

return;

// Try assigning the current worker to each task and recur for remaining workers

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

if (!assigned[task]) {

assigned[task] = 1;

branchAndBound(costMatrix, assigned, worker + 1, cost + costMatrix[worker][task], minCost);

assigned[task] = 0; // Backtrack

}

}

}

int main() {

int costMatrix[N][N];

int assigned[N] = {0}; // Array to keep track of task assignments

int minCost = INT\_MAX;

// Taking input for the cost matrix

printf("Enter the cost matrix (%d x %d):\n", N, N);

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

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

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

}

}

// Start from the first worker and compute the minimum cost using branch and bound

branchAndBound(costMatrix, assigned, 0, 0, &minCost);

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

return 0;

}

**Output:**

Enter the cost matrix (4 x 4):

0 1 2 3

1 2 3 4

2 3 4 5

3 4 5 6

Minimum cost of assignment: 12

--------------------------------

Process exited after 16.56 seconds with return value 0

Press any key to continue . . .

1. **Insert numbers in a list**

**Code:**

#include <stdio.h>

#include <stdlib.h>

// Define a structure for the node of the list

struct Node {

int data;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the end of the list

void insertNode(struct Node\*\* head, int data) {

if (\*head == NULL) {

\*head = createNode(data);

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = createNode(data);

}

}

// Function to display the list

void displayList(struct Node\* head) {

printf("List: ");

while (head != NULL) {

printf("%d ", head->data);

head = head->next;

}

printf("\n");

}

int main() {

int num;

char choice;

struct Node\* head = NULL; // Initialize an empty list

// Insert initial numbers into the list

printf("Enter initial numbers into the list (enter -1 to stop):\n");

while (1) {

scanf("%d", &num);

if (num == -1)

break;

insertNode(&head, num);

}

// Display the initial list

displayList(head);

// Insert a number into the list

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

scanf("%d", &num);

insertNode(&head, num);

// Display the updated list

printf("Updated ");

displayList(head);

// Free dynamically allocated memory to prevent memory leaks

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

return 0;

}

**Output:**

Enter initial numbers into the list (enter -1 to stop):

5

4 12 78 45 03

-1

List: 5 4 12 78 45 3

Enter a number to insert: 13

Updated List: 5 4 12 78 45 3 13

--------------------------------

Process exited after 26.85 seconds with return value 0

Press any key to continue . . .

1. **Optimal cost**

**Code:**

#include <stdio.h>

#define MAX\_ITEMS 100

// Function to find the maximum of two integers

int max(int a, int b) {

return (a > b) ? a : b;

}

// Function to find the maximum profit using dynamic programming

int knapsack(int weights[], int profits[], int n, int capacity) {

int dp[MAX\_ITEMS + 1][capacity + 1];

// Initialize dp table

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

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

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

dp[i][w] = 0;

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

dp[i][w] = max(profits[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

else

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

}

}

return dp[n][capacity];

}

int main() {

int weights[MAX\_ITEMS], profits[MAX\_ITEMS];

int n, capacity;

// Input the number of items

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

scanf("%d", &n);

// Input weights and profits of each item

printf("Enter weights and profits of each item:\n");

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

printf("Item %d: ", i + 1);

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

}

// Input the capacity of the knapsack

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

scanf("%d", &capacity);

// Find the maximum profit

int maxProfit = knapsack(weights, profits, n, capacity);

printf("Maximum profit: %d\n", maxProfit);

return 0;

}

**Output:**

Enter the number of items: 5

Enter weights and profits of each item:

Item 1: 3 10

Item 2: 3 15

Item 3: 2 10

Item 4: 5 12

Item 5: 1 8

Enter the capacity of the knapsack: 10

Maximum profit: 43

--------------------------------

Process exited after 30.28 seconds with return value 0

Press any key to continue . . .