192225004

CSA0652- Design and Analysis of Algorithms for AI Tools

1. Fibonacci series upto n 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;
  printf("Enter the number of terms: ");
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    printf("%d ", fibonacci(i));}
  return 0;}
```

2. Armstrong number using recursion

```
#include <stdio.h>
#include <math.h>
int digits(int n) {
  if (n == 0) return 0;
  return 1 + digits(n / 10);}
int armstrong(int n, int pow, int sum) {
  if (n == 0) return sum;
  return armstrong(n / 10, pow, sum + powl(n % 10, pow));}
int main() {
  int num;
  printf("Enter a number: ");
  scanf("%d", &num);
  int pow = digits(num);
  if (num == armstrong(num, pow, 0)) printf("Armstrong");
  else printf("Not Armstrong");
  return 0;}
```

3. GCD of two numbers using recursion

```
#include <stdio.h>
int gcd(int a, int b) {
  if (b == 0) return a;
```

```
    return gcd(b, a % b);}
int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("GCD: %d", gcd(a, b));
    return 0;}
```

4. Largest element in an array

```
#include <stdio.h>
int largest(int arr[], int n) {
    if (n == 1) return arr[0];
    int max = largest(arr, n - 1);
    return (arr[n - 1] > max) ? arr[n - 1] : max;}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);}
    printf("Largest: %d", largest(arr, n));
    return 0;}
```

5. Factorial of a number using recursion

```
#include <stdio.h>
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);}
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Factorial: %d", factorial(n));
    return 0;}
```

6. Check if a number is prime or not using recursion

```
#include <stdio.h>
int isPrime(int n, int i) {
    if (n <= 2) return (n == 2) ? 1 : 0;
    if (n % i == 0) return 0;
    if (i * i > n) return 1;
    return isPrime(n, i + 1);}
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (isPrime(n, 2)) printf("Prime");
    else printf("Not Prime");
    return 0;}
```

7. Selection Sort

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) minIndex = j;}
        int 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: ");
    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;}
```

8. Bubble sort

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int 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: ");
    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;}
```

9. Palindrome

```
#include <stdio.h>
#include <string.h>
int isPalindrome(char str[], int start, int end) {
  if (start >= end) return 1;
  if (str[start] != str[end]) return 0;
  return isPalindrome(str, start + 1, end - 1);}
int main() {
  char n[100];
  printf("Enter a number: ");
  scanf("%s", n);
  int len = strlen(n);
  if (isPalindrome(n, 0, len - 1)) printf("Palindrome");
  else printf("Not Palindrome");
  return 0;}
```

10. Time Complexity of multiplying two matrices

```
#include <stdio.h>
#include <math.h>
int is_power_of_two(int n) {
  return (n > 0) && ((n & (n - 1)) == 0);}
int main() {
  int n;
  double complexity;
  printf("Enter the size n for an n x n matrix: ");
  scanf("%d",&n);
```

```
    if (is_power_of_two(n)) {
      complexity = pow(n, 2.81);
      printf("Time Complexity of multiplying two %d x %d matrices: %.2f\n", n,n,complexity);}
          else {
      printf("Theoretical Value of Time Complexity = O(n^log2(7))\n");}
    return 0;}
```

11. Program to copy one string to another

```
#include <stdio.h>
int main() {
    char str1[100], str2[100];
    printf("Enter a string: ");
    scanf("%s", str1);
    int i = 0;
    while (str1[i] != '\0') {
      str2[i] = str1[i];
      i++;}
    str2[i] = '\0';
    printf("Copied string: %s\n", str2);
    return 0;}
```

12. Program to perform binary search

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x) {
    while (l <= r) {
      int m = l + (r - l) / 2;
      if (arr[m] == x)  return m;
      if (arr[m] < x)  l = m + 1;
      else  r = m - 1;}
    return -1;}
int main() {
    int n, x, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter elements: ");
    for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
    printf("Enter element to search: ");
    scanf("%d", &x);
    int result = binarySearch(arr, 0, n-1, x);
    if (result == -1)  printf("Element not found\n");
    else  printf("Element found at index %d\n", result);
    return 0;}
```

13. Reverse a string

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int n = strlen(str);
    for (int i = 0; i < n / 2; i++) {
        char temp = str[i];
        str[i] = str[n - i - 1];
        str[n - i - 1] = temp;}
    printf("Reversed string: %s\n", str);
    return 0;}
```

14. Find length of a string

```
#include <stdio.h>
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int length = 0;
    while (str[length] != '\0')
        length++;
    printf("Length of the string: %d\n", length);
    return 0;}
```

15. Perform strassen’s matrix multiplication

```
#include<stdio.h>
int main(){
  int z[2][2];
  int i, j;
  int m1, m2, m3, m4 , m5, m6, m7;
  int A[2][2] = { {12, 34}, {22, 10}};
  int B[2][2] = { {3, 4}, {2, 1}};
  printf("Matrix 1: ");
  for(i = 0; i < 2; i++) {
    printf("\n");
    for(j = 0; j < 2; j++)
      printf("%d\t", A[i][j]);}
  printf("\nMatrix 2: ");
  for(i = 0; i < 2; i++) {
    printf("\n");
    for(j = 0; j < 2; j++)
      printf("%d\t", B[i][j]);}
  m1= (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
  m2= (A[1][0] + A[1][1]) * B[0][0];
  m3= A[0][0] * (B[0][1] - B[1][1]);
  m4= A[1][1] * (B[1][0] - B[0][0]);
  m5= (A[0][0] + A[0][1]) * B[1][1];
  m6= (A[1][0] - A[0][0]) * (B[0][0]+B[0][1]);
  m7= (A[0][1] - A[1][1]) * (B[1][0]+B[1][1]);
  z[0][0] = m1 + m4- m5 + m7;
  z[0][1] = m3 + m5;
  z[1][0] = m2 + m4;
  z[1][1] = m1 - m2 + m3 + m6;
  printf("\nProduct of Strassen's Matrix Multiplication: ");
  for(i = 0; i < 2 ; i++) {
    printf("\n");
    for(j = 0; j < 2; j++)
      printf("%d\t", z[i][j]);}
  return 0;}
```

16. Perform merge sort

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[m + 1 + i];
    int i = 0, j = 0, k = l;
    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 l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);}}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter 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;}
```

17. Use divide and conquer to find max and min value in the list

```
#include <stdio.h>
void findMinMax(int arr[], int low, int high, int *min, int *max) {
    if (low == high) {
        *min = arr[low];
        *max = arr[low];}
            else if (low == high - 1) {
        if (arr[low] < arr[high]) {
            *min = arr[low];  *max = arr[high];}
                        else {
            *min = arr[high];  *max = arr[low];}}
            else {
        int mid = (low + high) / 2;
        int min1, max1, min2, max2;
        findMinMax(arr, low, mid, &min1, &max1);
        findMinMax(arr, mid + 1, high, &min2, &max2);
        *min = (min1 < min2) ? min1 : min2;
        *max = (max1 > max2) ? max1 : max2;}}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    int min, max;
    findMinMax(arr, 0, n - 1, &min, &max);
    printf("Minimum element: %d\n", min);
    printf("Maximum element: %d\n", max);
    return 0;}
```

18. Generate all prime numbers upto n

```
#include <stdio.h>
#include <stdbool.h>
void generatePrimes(int n) {
    bool prime[n + 1];
    for (int i = 0; i <= n; i++)
        prime[i] = true;
    for (int p = 2; p * p <= n; p++) {
        if (prime[p] == true) {
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;}}
    for (int p = 2; p <= n; p++)
        if (prime[p])
            printf("%d ", p);
    printf("\n");}
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    generatePrimes(n);
    return 0;}
```

19. Perform knapsack problem using greedy approach

```
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0;
   int i, j, u;
```

```
  u = capacity;
  for (i = 0; i < n; i++)
    x[i] = 0.0;
  for (i = 0; i < n; i++) {
    if (weight[i] > u)
      break;
    else {
      x[i] = 1.0;
      tp = tp + profit[i];
      u = u - weight[i];}}
  if (i < n)
    x[i] = u / weight[i]; tp = tp + (x[i] * profit[i]);
  printf("\nThe result vector is: ");
  for (i = 0; i < n; i++)
    printf("%f\t", x[i]);
  printf("\nMaximum profit is: %f", tp);}
int main() {
  float weight[20], profit[20], capacity, ratio[20], temp;
  int num, i, j;
  printf("\nEnter the no. of objects:- ");
  scanf("%d", &num);
  printf("\nEnter the weights and profits of each object: ");
  for (i = 0; i < num; i++) {
    scanf("%f %f", &weight[i], &profit[i]);}
  printf("\nEnter the capacity of knapsack: ");
  scanf("%f", &capacity);
  for (i = 0; i < num; i++) {
    ratio[i] = profit[i] / weight[i];}
  for (i = 0; i < num; i++) {
    for (j = i + 1; j < num; j++) {
      if (ratio[i] < ratio[j]) {
        temp = ratio[j];
        ratio[j] = ratio[i];
        ratio[i] = temp;
        temp = weight[j];
        weight[j] = weight[i];
        weight[i] = temp;
        temp = profit[j];
        profit[j] = profit[i];
        profit[i] = temp;}}}
  knapsack(num, weight, profit, capacity);
  return(0);}
```

20. Perform minimum spanning tree using greedy approach

```
#include <stdio.h>
#include <limits.h>
#define V 5
int minKey(int key[], int mstSet[]) {
  int min = INT_MAX, min_index;
  int v;
  for (v = 0; v < V; v++)
    if (mstSet[v] == 0 && key[v] < min)
      min = key[v], min_index = v;
  return min_index;}
int printMST(int parent[], int n, int graph[V][V]) {
  int i;
  printf("Edge   Weight\n");
  for (i = 1; i < V; i++)
    printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);}
void primMST(int graph[V][V]) {
  int parent[V];
  int key[V], i, v, count;
  int mstSet[V];
  for (i = 0; i < V; i++){
    key[i] = INT_MAX, mstSet[i] = 0; key[0] = 0; parent[0] = -1; }
  for (count = 0; count < V - 1; count++) {
    int u = minKey(key, mstSet); mstSet[u] = 1;
    for (v = 0; v < V; v++){
      if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
        parent[v] = u, key[v] = graph[u][v];}}// print the constructed MST
  printMST(parent, V, graph);}
int main() {
  int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },
      { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };
  primMST(graph);
  return 0;}
```

21. Dynamic programming OBST

```
#include <stdio.h>
#include <limits.h>
int sum(int freq[], int low, int high) {
  int sum = 0;
  for (int k = low; k <= high; k++)
    sum += freq[k];
  return sum;}
```

```
int minCostBST(int keys[], int freq[], int n) {
  int cost[n][n];
  for (int i = 0; i < n; i++)
    cost[i][i] = freq[i];
  for (int length = 2; length <= n; length++) {
    for (int i = 0; i <= n - length; i++) {
      int j = i + length - 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 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", minCostBST(keys, freq, n));
  return 0;}
```

22. Binomial Coefficient using Dynamic programming

```
#include <stdio.h>
int binomialCoeff(int n, int k) {
  int C[n + 1][k + 1];
  int i, j;
  for (i = 0; i <= n; i++) {
    for (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, k;
  printf("Enter n: ");
  scanf("%d", &n);
  printf("Enter k: ");
  scanf("%d", &k);
  printf("Binomial Coefficient C(%d, %d) is %d\n", n, k, binomialCoeff(n, k));
  return 0;}
```

23. Reverse of a given number

```
#include <stdio.h>
int main() {
   int num, reversed = 0;
   printf("Enter a number: ");
   scanf("%d", &num);
   while (num != 0) {
      reversed = reversed * 10 + num % 10;
      num /= 10;}
   printf("Reversed number is: %d\n", reversed);
   return 0;}
```

24. Check if number is perfect number

```
#include <stdio.h>
int main() {
   int num, sum = 0;
   printf("Enter a number: ");
   scanf("%d", &num);
   for (int i = 1; i <= num / 2; i++) {
      if (num % i == 0) {
         sum += i;}}
   if (sum == num && num != 0) {
      printf("Perfect number.\n", num);}
   else { printf("Not a perfect number.\n", num);}
   return 0;}
```

25. Pascal triangle

```
#include <stdio.h>
int main() {
   int n;
   printf("Enter number of rows: ");
   scanf("%d", &n);
   for (int line = 0; line < n; line++) {
      int value = 1;
      for (int i = 1; i <= n - line; i++)
         printf(" ");
      for (int i = 0; i <= line; i++) {
         printf("%d ", value);
         value = value * (line - i) / (i + 1);}
      printf("\n");}
   return 0;}
```

26. Sum of digits

```
#include <stdio.h>
int main() {
  int num, sum = 0;
  printf("Enter a number: ");
  scanf("%d", &num);
  while (num != 0) {
    sum += num % 10;
    num /= 10;}
  printf("Sum of digits is: %d\n", sum);
  return 0;}
```

27. Pattern

```
#include <stdio.h>
int main() {
  int n;
  printf("Enter the number of rows: ");
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n - i; j++) { printf("  ");}
    for (int j = 1; j <= i; j++) { printf("%d ", j);}
    printf("\n");}
  return 0;}
```

28. Travelling Salesman using Dynamic Programming

```
#include <stdio.h>
#include <limits.h>
#define MAX 20
#define INF INT_MAX
int n;
int dist[MAX][MAX];
int dp[MAX][1 << MAX];
int tsp(int pos, int mask) {
  if (mask == ((1 << n) - 1)) { return dist[pos][0];}
  if (dp[pos][mask] != -1) { return dp[pos][mask];}
  int ans = INF;
  for (int city = 0; city < n; city++) {
    if ((mask & (1 << city)) == 0) {
      int newAns = dist[pos][city] + tsp(city, mask | (1 << city));
      if (newAns < ans) { ans = newAns;}}}
  return dp[pos][mask] = ans;}
int main() {
  printf("Enter the number of cities: ");
  scanf("%d", &n);
  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]);}
  for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < (1 << MAX); j++) { dp[i][j] = -1;}
  printf("The minimum cost of visiting all cities: %d\n", tsp(0, 1));
  return 0;}
```

29. Floyd's algorithm

```
#include <stdio.h>
#define INF 99999
#define MAX 100
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++) {
            dist[i][j] = graph[i][j];}}
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];}}}}
    printf("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("%7s", "INF");}
                                else { printf("%7d", dist[i][j]);}}
        printf("\n");}}
int main() {
    int n;
    int graph[MAX][MAX];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (use %d to represent infinity):\n", INF);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);}}
    floydWarshall(graph, n);
    return 0;}
```

30. Optimal Cost using appropriate algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define V 6
int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++) {
        if (sptSet[v] == 0 && dist[v] <= min) {
            min = dist[v];
            min_index = v;}}
    return min_index;}
void printSolution(int dist[], int n) {
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < n; i++)
        printf("%d \t\t %d\n", i, dist[i]);}
void dijkstra(int graph[V][V], int src, int dest) {
    int dist[V], sptSet[V];
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX; sptSet[i] = 0;}
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = 1;
        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];}}
    printSolution(dist, V);
    printf("Optimal Cost from node %d to node %d: %d\n", src, dest, dist[dest]);}
int main() {
    int graph[V][V] = { {0, 4, 0, 0, 0, 0}, {4, 0, 8, 0, 0, 0}, {0, 8, 0, 7, 0, 4},
        {0, 0, 7, 0, 9, 14},{0, 0, 4, 14, 10, 0}};
    int source = 0, destination = 5;
    dijkstra(graph, source, destination);
    return 0;}
```

31. Minimum to Maximum value sequence

```
#include <stdio.h>
int findMin(int arr[], int size) {
    int min = arr[0];
    for(int i = 1; i < size; i++) {
        if(arr[i] < min) { min = arr[i];}}
    return min;}
int findMax(int arr[], int size) {
    int max = arr[0];
    for(int i = 1; i < size; i++) { if(arr[i] > max) { max = arr[i];}}
    return max;}
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]);}
    int min = findMin(arr, n);
    int max = findMax(arr, n);
    printf("Sequence from minimum to maximum value:\n");
    for(int i = min; i <= max; i++) {
        printf("%d ", i);}
    return 0;}
```

32. N-queens using backtracking

```
#include <stdio.h>
#include <stdbool.h>
#define N 4
void printBoard(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", board[i][j]);}
        printf("\n");}}
bool isSafe(int board[N][N], int row, int col) {
    int i, j;
    for (i = 0; i < col; i++) {
        if (board[row][i]) { return false;}}
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;}}
    for (i = row, j = col; i < N && j >= 0; i++, j--) {
        if (board[i][j]) { return false;}}
    return true;}
bool solveNQueensUtil(int board[N][N], int col) {
    if (col >= N) { return true;}
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQueensUtil(board, col + 1)) { return true;}
            board[i][col] = 0; }}
                                return false;}
bool solveNQueens() {
    int board[N][N] = {0};
    if (!solveNQueensUtil(board, 0)) {
        printf("Solution does not exist\n");
        return false;}
    printBoard(board);
    return true;}
int main() { solveNQueens(); return 0;}
```

33. Insert element

```
#include <stdio.h>
int main() {
    int arr[100], n, element, position;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n", n);
    for (int i = 0; i < n; i++)
            { scanf("%d", &arr[i]);}
    printf("Enter the element to be inserted: ");
    scanf("%d", &element);
    printf("Enter the position to insert new element: ");
    scanf("%d", &position);
    if (position < 0 || position > n) {
        printf("Invalid position." );}
            else {
        for (int i = n; i > position; i--) {
            arr[i] = arr[i - 1];}
        arr[position] = element;
        n++;
        printf("Array after insertion:\n");
        for (int i = 0; i < n; i++) {
            printf("%d ", arr[i]);}
        printf("\n");}
    return 0;}
```

34. Sum of subsets

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
void findSubsets(int arr[], int n, int sum, int subset[], int subsetSize, int currentIndex, int
currentSum) {
  if (currentSum == sum) {
    printf("{ ");
    for (int i = 0; i < subsetSize; i++) {
      printf("%d ", subset[i]);}
    printf("}\n");
    return;}
  if (currentIndex >= n || currentSum > sum) { return;}
  subset[subsetSize] = arr[currentIndex];
  findSubsets(arr, n, sum, subset, subsetSize + 1, currentIndex + 1, currentSum +
arr[currentIndex]);
  findSubsets(arr, n, sum, subset, subsetSize, currentIndex + 1, currentSum);}
int main() {
  int n, sum;
  int arr[MAX_SIZE];
  printf("Enter the size of the set: ");
  scanf("%d", &n);
  printf("Enter %d elements:\n", n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);}
  printf("Enter the sum value: ");
  scanf("%d", &sum);
  int subset[MAX_SIZE];
  printf("Subsets with sum %d :\n", sum);
  findSubsets(arr, n, sum, subset, 0, 0, 0);
  return 0;}
```

35. Graph Coloring

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
struct Graph {
  int V, adj[MAX_VERTICES][MAX_VERTICES];};
void initGraph(struct Graph *G, int V) {
  G->V = V;
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) { G->adj[i][j] = 0;}}}
```

```
void addEdge(struct Graph *G, int u, int v) {
  G->adj[u][v] = 1; G->adj[v][u] = 1; }
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]);}}
bool isSafe(struct Graph *G, int v, int color[], int c) {
  for (int i = 0; i < G->V; i++) {
    if (G->adj[v][i] && c == color[i]) { return false;}}
  return true;}
void graphColoring(struct Graph *G, int m) {
  int color[MAX_VERTICES];
  for (int i = 0; i < G->V; i++) { color[i] = 0;}
  for (int v = 1; v < G->V; v++) {
    for (int c = 1; c <= m; c++) {
      if (isSafe(G, v, color, c)) {
        color[v] = c;
        break;}}}
  printSolution(color, G->V);}
int main() {
  struct Graph G;
  int V, E;
  printf("Enter the number of vertices: ");
  scanf("%d", &V);
  printf("Enter the number of edges: ");
  scanf("%d", &E);
  initGraph(&G, V);
  printf("Enter %d edges (format: u v):\n", E);
  for (int i = 0; i < E; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    addEdge(&G, u, v);}
  int m;
  printf("Enter the number of colors: ");
  scanf("%d", &m);
  graphColoring(&G, m);
  return 0;}
```

36. Container Loading

```
#include <stdio.h>
#define MAX_ITEMS 100
void containerLoading(int numItems, int weights[], int capacity) {
  int used[MAX_ITEMS] = {0}, remaining_capacity = capacity, max_weight_index = 0;
```

```
    for (int i = 1; i < numItems; i++) {
      if (weights[i] > weights[max_weight_index]) {
        max_weight_index = i;}}
    for (int i = max_weight_index; i < numItems; i++) {
      if (weights[i] <= remaining_capacity) {
        used[i] = 1;
        remaining_capacity -= weights[i];}}
    printf("Output:\n");
    for (int i = 0; i < numItems; i++) {
      printf("%d ", used[i]);}
    printf("\n");}
  int main() {
    int numItems, capacity, weights[MAX_ITEMS];
    printf("Enter number of items: ");
    scanf("%d", &numItems);
    printf("Enter weights of each item:\n");
    for (int i = 0; i < numItems; i++) {
      scanf("%d", &weights[i]);}
    printf("Enter maximum capacity of the container: ");
    scanf("%d", &capacity);
    containerLoading(numItems, weights, capacity);
    return 0;}
```

37. List of all factors of n

```
#include <stdio.h>
int main() {
  int n;
  printf("Enter an integer: ");
  scanf("%d", &n);
  printf("Factors of %d are: ", n);
  for (int i = 1; i <= n; ++i) {
    if (n % i == 0) { printf("%d ", i);}}
  printf("\n");
  return 0;}
```

38. Assignment Problem using Branch and Bound

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#define MAXN 100
int n, cost[MAXN][MAXN], result[MAXN], match[MAXN];
bool assigned[MAXN];
```

```
void initialize() {
    for (int i = 0; i < n; ++i) {
        result[i] = -1;
        assigned[i] = false;
        match[i] = -1;}}
bool canAssign(int worker, int task) {
    return !assigned[task] && (cost[worker][task] == 0);}
bool tryAssign(int worker) {
    for (int task = 0; task < n; ++task) {
        if (canAssign(worker, task)) {
            assigned[task] = true;
            if (match[task] == -1 || tryAssign(match[task])) {
                match[task] = worker;
                return true;}}}
    return false;}
void Assignment() {
    initialize();
    for (int worker = 0; worker < n; ++worker) {
        while (true) {
            int min_cost = INT_MAX;
            for (int task = 0; task < n; ++task) {
                if (!assigned[task] && cost[worker][task] < min_cost) {
                    min_cost = cost[worker][task];}}
            for (int task = 0; task < n; ++task) {
                if (!assigned[task]) {
                    cost[worker][task] -= min_cost;}}
            if (tryAssign(worker)) {break;}
            for (int task = 0; task < n; ++task) {
                if (assigned[task]) { cost[worker][task] += min_cost;}}}}
    for (int task = 0; task < n; ++task) {
        result[match[task]] = task;}}
void printAssignment() {
    printf("Worker   Task\n");
    for (int i = 0; i < n; ++i) {
        printf("%d       %d\n", i, result[i]);}}
int main() {
    printf("Enter number of workers and tasks (n): ");
    scanf("%d", &n);
    printf("Enter cost matrix (%dx%d):\n", n, n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &cost[i][j]);}}
    Assignment();
    printf("Optimal Assignment:\n");
```

```
  printAssignment();
  return 0;}
```

39. Linear Search

```
#include <stdio.h>
int linearSearch(int arr[], int n, int target) {
  for (int i = 0; i < n; ++i) {
    if (arr[i] == target) { return i;}}
  return -1; }
int main() {
  int n, target;
  printf("Enter number of elements in the array: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements:\n", n);
  for (int i = 0; i < n; ++i) {
    scanf("%d", &arr[i]);}
  printf("Enter the target element to search: ");
  scanf("%d", &target);
  int index = linearSearch(arr, n, target);
  if (index != -1) { printf("Target element found at index %d.\n", target, index);}
        else { printf("Not found in the array.\n", target);}
  return 0;}
```

40. Hamiltonian Circuit

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
int n, graph[MAX_VERTICES][MAX_VERTICES], path[MAX_VERTICES];
bool isSafe(int v, int pos, int path[]) {
  if (graph[path[pos - 1]][v] == 0) { return false;}
  for (int i = 0; i < pos; ++i) {
    if (path[i] == v) {
      return false;}}
  return true;}
bool hamiltonianCircuitUtil(int path[], int pos) {
  if (pos == n) {
    if (graph[path[pos - 1]][path[0]] == 1) { return true; }
                 else { return false; }}
  for (int v = 1; v < n; ++v) {
    if (isSafe(v, pos, path)) {
            path[pos] = v;
            if (hamiltonianCircuitUtil(path, pos + 1)) {return true;}
            path[pos] = -1;}}
    return false; }
void hamiltonianCircuit() {
    int path[MAX_VERTICES];
    for (int i = 0; i < n; ++i) {
        path[i] = -1;}
    path[0] = 0;
    if (hamiltonianCircuitUtil(path, 1) == false) {
        printf("No Hamiltonian Circuit exists.\n");
        return;}
    printf("Hamiltonian Circuit found: ");
    for (int i = 0; i < n; ++i) { printf("%d ", path[i]);}
    printf("%d\n", path[0]); }
int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    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]);}}
    hamiltonianCircuit();
    return 0;}
```