CSA0630
DESIGN ANALYSIS AND ALGORITHMS FOR SORTING
PRACTICAL SESSION DAY 3

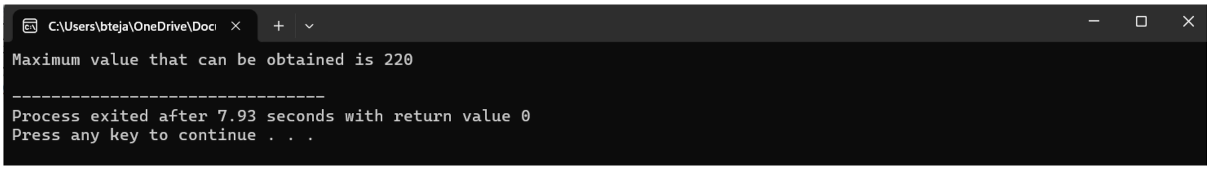
1.Write a c program for knapsack problem using dynamic programming

PROGRAM:

```
#include <stdio.h>

int max(int a, int b) {

   return (a > b) ? a : b;

}

int knapsack(int W, int wt[], int val[], int n) {

   int i, w;

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

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

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

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

            dp[i][w] = 0;

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

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

         else

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

      }

   }



   return dp[n][W];

}
```

```
int main() {
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    int result = knapsack(W, wt, val, n);

    printf("Maximum value that can be obtained is %d\n", result);

    return 0;
}
```

OUTPUT:



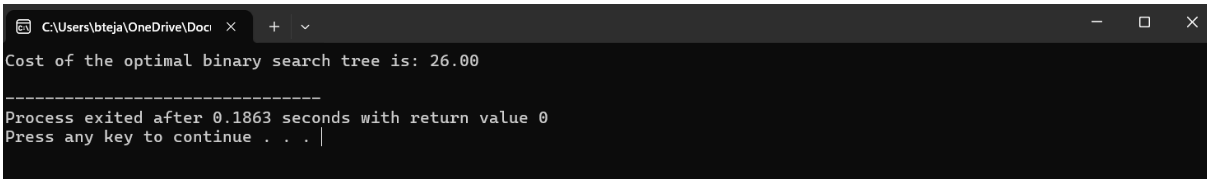
```
Maximum value that can be obtained is 220

--------------------------------
Process exited after 7.93 seconds with return value 0
Press any key to continue . . .
```

2.Using Dynamic programming concept to find out Optimal binary search tree

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
float sum(float freq[], int i, int j) {
    float s = 0;
    for (int k = i; k <= j; k++)
        s += freq[k];
    return s;
}
float optimalBST(float keys[], float freq[], int n) {
    float cost[n][n];
    for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];
    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len + 1; i++) {
            int j = i + len - 1;
            cost[i][j] = INT_MAX;
            for (int r = i; r <= j; r++) {
                float 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() {
    float keys[] = {10, 12, 16, 21};
    float freq[] = {4, 2, 6, 3};
    int n = sizeof(keys) / sizeof(keys[0]);

    float result = optimalBST(keys, freq, n);

    printf("Cost of the optimal binary search tree is: %.2f\n", result);

    return 0;

}
```

OUTPUT:



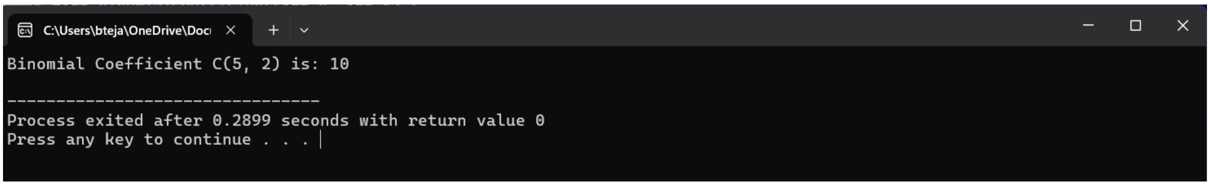
```
Cost of the optimal binary search tree is: 26.00

--------------------------------
Process exited after 0.1863 seconds with return value 0
Press any key to continue . . .
```

3.Using Dynamic programming techniques to find binomial coefficient of a given number

PROGRAM:

```
#include <stdio.h>
int binomialCoefficient(int n, int k) {
    int dp[n + 1][k + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= k && j <= i; j++) {
            if (j == 0 || j == i)
                dp[i][j] = 1;
            else
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
        }
    }
    return dp[n][k];
}
int main() {
    int n = 5, k = 2;
    int result = binomialCoefficient(n, k);
    printf("Binomial Coefficient C(%d, %d) is: %d\n", n, k, result);
    return 0;
}
```

OUTPUT:



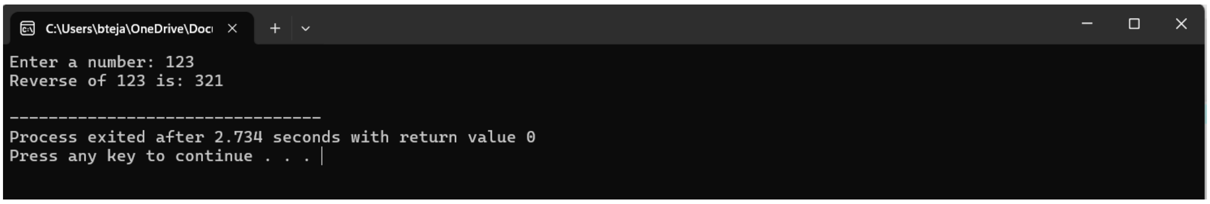
```
Binomial Coefficient C(5, 2) is: 10

--------------------------------
Process exited after 0.2899 seconds with return value 0
Press any key to continue . . .
```

4.Write a program to find the reverse of a given number using recursive

PROGRAM:

```
\#include <iostream>
#include <cmath>
int reverseNumber(int num) {
    if (num < 10) {
        return num;
    } else {
        return (num % 10) * std::pow(10, static_cast<int>(std::log10(num))) +
reverseNumber(num / 10);
    }
}
int main() {
    int num;
    std::cout << "Enter a number: ";
    std::cin >> num;
    int reversedNum = reverseNumber(num);
    std::cout << "Reverse of " << num << " is: " << reversedNum << std::endl;

    return 0;
}
```

OUTPUT:



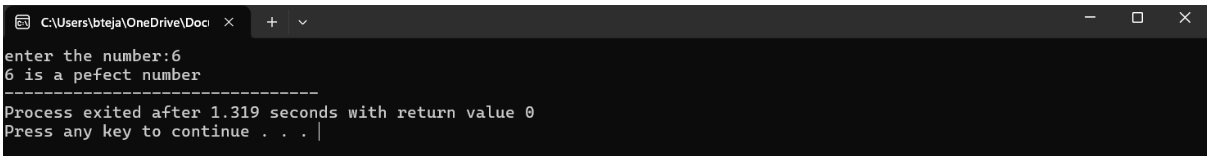
```
Enter a number: 123
Reverse of 123 is: 321

--------------------------------
Process exited after 2.734 seconds with return value 0
Press any key to continue . . .
```

5.Write a program to find the perfect number.

PROGRAM:

```
#include<stdio.h>
int main()
{
        int n,i,sum=0,rem;
        printf("enter the number:");
        scanf("%d",&n);
        for(i=1;i<n;i++)
        {
          rem=n%i;
          if(rem==0)
          {
                 sum=sum+i;
          }
        }
        if(sum==n)
                printf("%d is a pefect number",n);
        else
                printf("%d is not a perfect number:",n);
        return 0;
}
```

OUTPUT:



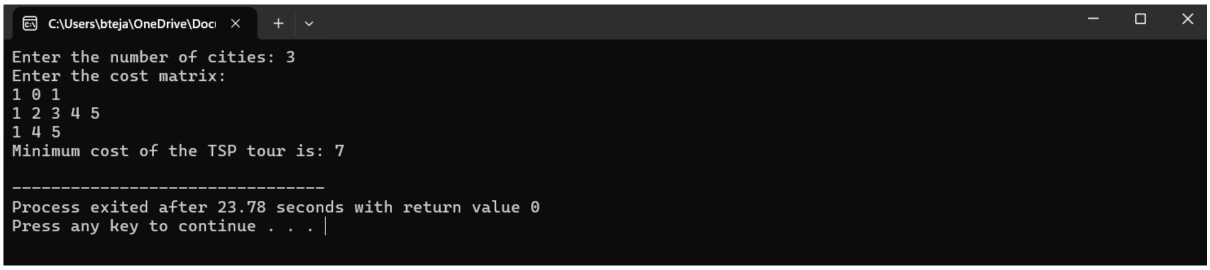
```
enter the number:6
6 is a pefect number
--------------------------------
Process exited after 1.319 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

```
#include <stdio.h>
#include <limits.h>

#define MAX 10

int memo[MAX][1 << MAX];
int tsp(int graph[MAX][MAX], int n, int mask, int pos) {
    if (mask == (1 << n) - 1) {
        return graph[pos][0];
    }

    if (memo[pos][mask] != -1) {
        return memo[pos][mask];
    }

    int minCost = INT_MAX;

    for (int i = 0; i < n; i++) {
        if ((mask & (1 << i)) == 0) {
            int newCost = graph[pos][i] + tsp(graph, n, mask | (1 << i), i);

            if (newCost < minCost) {
                minCost = newCost;
            }
        }
    }

    return memo[pos][mask] = minCost;
}
int main() {
    int n, i, j;

    printf("Enter the number of cities: ");
    scanf("%d", &n);

    int graph[MAX][MAX];

    printf("Enter the cost matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    for (i = 0; i < MAX; i++) {
        for (j = 0; j < (1 << MAX); j++) {
            memo[i][j] = -1;
        }
    }

    int start = 0;



    int minCost = tsp(graph, n, 1 << start, start);

    printf("Minimum cost of the TSP tour is: %d\n", minCost);

    return 0;
}
```

OUTPUT:



```
Enter the number of cities: 3
Enter the cost matrix:
1 0 1
1 2 3 4 5
1 4 5
Minimum cost of the TSP tour is: 7

--------------------------------
Process exited after 23.78 seconds with return value 0
Press any key to continue . . .
```

7.Write a program for the given pattern using recursion

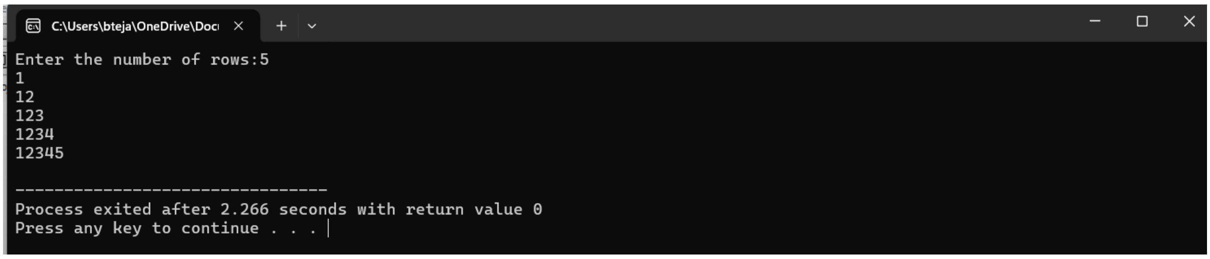
If n=4

```
1
1  2
1  2   3
1  2   3   4
```

PROGRAM:

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

OUTPUT:



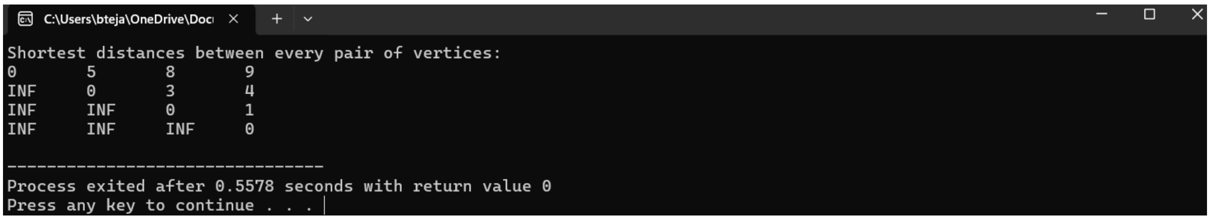
```
Enter the number of rows:5
1
12
123
1234
12345

--------------------------------
Process exited after 2.266 seconds with return value 0
Press any key to continue . . .
```

8. Write a program to perform Floyd’s algorithm

PROGRAM:

```
#include <stdio.h>
#define INF 9999
#define V 4
void floydWarshall(int graph[V][V]) {
    int dist[V][V];
    int i, j, k;
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
        }
    }
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    printf("Shortest distances between every pair of vertices:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][j] == INF) {
                printf("INF\t");
            } else {
                printf("%d\t", dist[i][j]);
            }
        }
        printf("\n");
    }
}
int main() {
    int graph[V][V] = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };
    floydWarshall(graph);

    return 0;
}
```

OUTPUT:



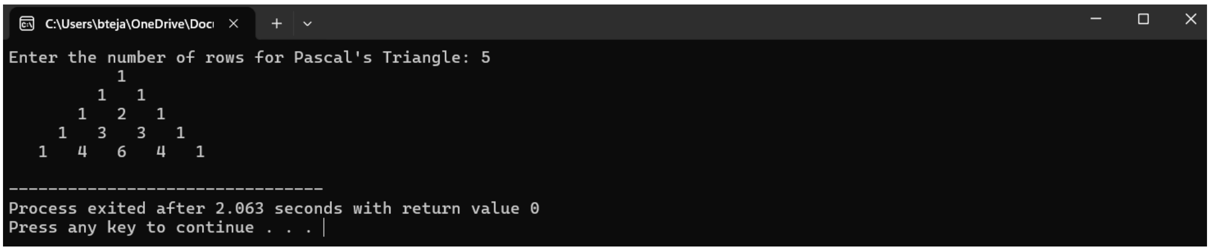
```
Shortest distances between every pair of vertices:
0       5       8       9
INF     0       3       4
INF     INF     0       1
INF     INF     INF     0

--------------------------------
Process exited after 0.5578 seconds with return value 0
Press any key to continue . . .
```

9. Write a program for pascal triangle

PROGRAM:

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
void generatePascalTriangle(int numRows) {
    for (int i = 0; i < numRows; i++) {
        for (int space = 0; space < numRows - i - 1; space++) {
            printf("  ");
        }

        for (int j = 0; j <= i; j++) {
            int coefficient = factorial(i) / (factorial(j) * factorial(i - j));
            printf("%4d", coefficient);
        }
        printf("\n");
    }
}
int main() {
    int numRows;
    printf("Enter the number of rows for Pascal's Triangle: ");
    scanf("%d", &numRows);
    generatePascalTriangle(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 2.063 seconds with return value 0
Press any key to continue . . .
```

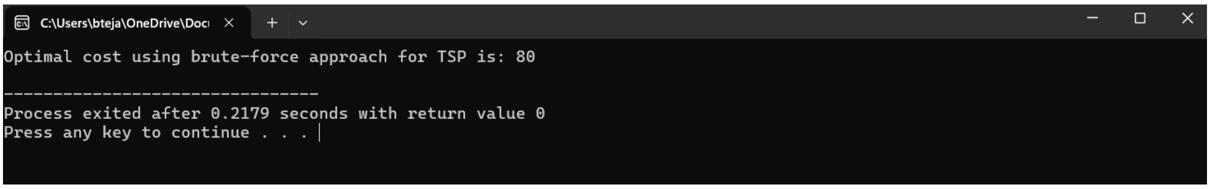
10. Write a program to find the optimal cost by using appropriate algorithm

PROGRAM:

```
\#include <stdio.h>

#include <limits.h>



#define V 4 // Number of vertices (cities)
```

```
int graph[V][V] = {
    {0, 10, 15, 20},
    {10, 0, 35, 25},
    {15, 35, 0, 30},
    {20, 25, 30, 0}
};

int min(int a, int b) {
    return (a < b) ? a : b;
}

// Function to find the optimal cost using brute-force approach
int tsp(int mask, int pos) {
    if (mask == (1 << V) - 1) {
        return graph[pos][0]; // Return to the starting city
    }

    int minCost = INT_MAX;

    for (int city = 0; city < V; city++) {
        if ((mask & (1 << city)) == 0) { // Check if the city has not been visited
            int newCost = graph[pos][city] + tsp(mask | (1 << city), city);
            minCost = min(minCost, newCost);
        }
    }

    return minCost;
}

// Driver program to test the function
int main() {
    int mask = 1; // Start with the first city
    int startCity = 0; // Starting city

    int result = tsp(mask, startCity);

    printf("Optimal cost using brute-force approach for TSP is: %d\n", result);

    return 0;
}
```

OUTPUT:



```
Optimal cost using brute-force approach for TSP is: 80

--------------------------------
Process exited after 0.2179 seconds with return value 0
Press any key to continue . . .
```