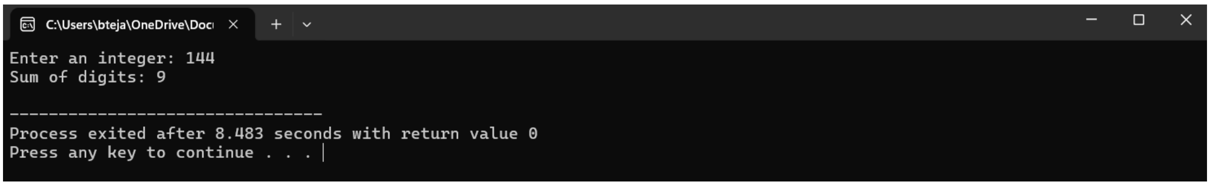
1. Write a program to find the sum of digits.

PROGRAM:

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

OUTPUT:



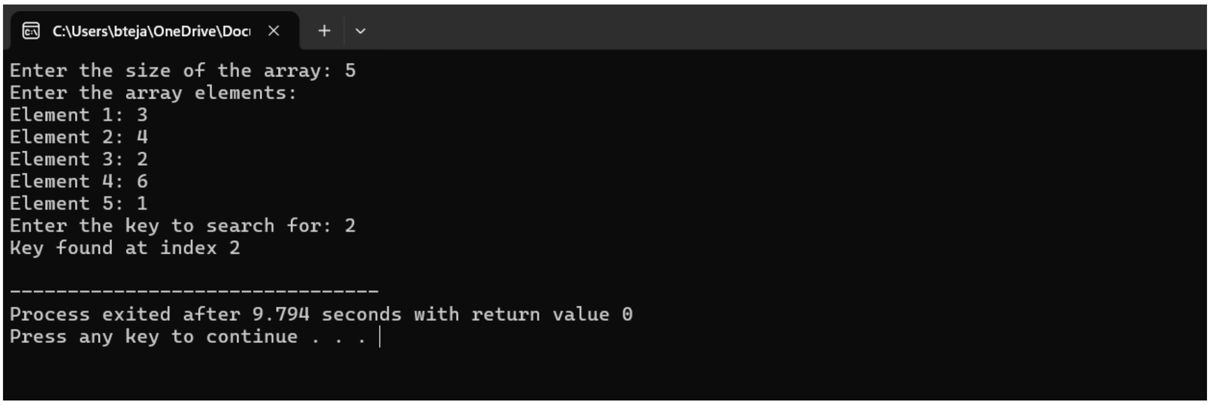
```
Enter an integer: 144
Sum of digits: 9

--------------------------------
Process exited after 8.483 seconds with return value 0
Press any key to continue . . .
```

2. Write a program for to perform liner search.

PROGRAM:

```
#include <stdio.h>
int linearSearch(const int arr[], int size, int key) {
    for (int i = 0; i < size; ++i) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}
int main() {
    const int maxSize = 100;
    int arr[maxSize];
    int size, key;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    printf("Enter the array elements:\n");
    for (int i = 0; i < size; ++i) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }
    printf("Enter the key to search for: ");
    scanf("%d", &key);
    int result = linearSearch(arr, size, key);
    if (result != -1) {
        printf("Key found at index %d\n", result);
    } else {
        printf("Key not found in the array.\n");
    }
    return 0;
}
```

OUTPUT:



```
Enter the size of the array: 5
Enter the array elements:
Element 1: 3
Element 2: 4
Element 3: 2
Element 4: 6
Element 5: 1
Enter the key to search for: 2
Key found at index 2

--------------------------------
Process exited after 9.794 seconds with return value 0
Press any key to continue . . .
```

3. Write a program to perform n Queens problem using backtracking.

PROGRAM:

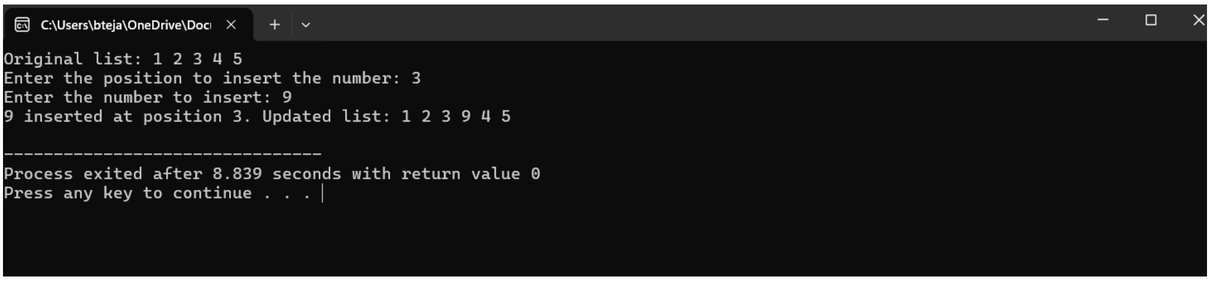
```
#include <stdio.h>
#include <stdlib.h>
void printBoard(char** board, int N) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            printf("%c ", board[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
int isSafe(char** board, int row, int col, int N) {
    for (int i = 0; i < col; ++i) {
        if (board[row][i] == 'Q') {
            return 0;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; --i, --j) {
        if (board[i][j] == 'Q') {
            return 0;
        }
    }
    for (int i = row, j = col; i < N && j >= 0; ++i, --j) {
        if (board[i][j] == 'Q') {
            return 0;
        }
    }
    return 1;
}
void solveNQueens(char** board, int col, int N) {
    if (col == N) {
        printBoard(board, N);
        return;
    }
    for (int i = 0; i < N; ++i) {
        if (isSafe(board, i, col, N)) {
            board[i][col] = 'Q';
            solveNQueens(board, col + 1, N);
            board[i][col] = '.';
        }
    }
}
int main() {
    int N;
    printf("Enter the size of the chessboard (N): ");
    scanf("%d", &N);
    char** board = (char*)malloc(N * sizeof(char));
    for (int i = 0; i < N; ++i) {
        board[i] = (char*)malloc(N * sizeof(char));
        for (int j = 0; j < N; ++j) {
            board[i][j] = '.';
        }
    }
    solveNQueens(board, 0, N);
    for (int i = 0; i < N; ++i) {
        free(board[i]);
    }
    free(board);
    return 0;
}
```

4. Write a program to inset a number in a list.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
void insertNumber(int* lst, int size, int position, int number) {
    if (position < 0 || position > size) {
        printf("Invalid position. Please choose a position between 0 and %d\n", size);
        return;
    }
    int* new_lst = (int*)malloc((size + 1) * sizeof(int));
    for (int i = 0; i < position; ++i) {
        new_lst[i] = lst[i];
    }
    new_lst[position] = number;
    for (int i = position + 1; i < size + 1; ++i) {
        new_lst[i] = lst[i - 1];
    }
    printf("%d inserted at position %d. Updated list: ", number, position);
    for (int i = 0; i < size + 1; ++i) {
        printf("%d ", new_lst[i]);
    }
    printf("\n");
    free(new_lst);
}
int main() {
    int originalList[] = {1, 2, 3, 4, 5};
    int size = sizeof(originalList) / sizeof(originalList[0]);
    printf("Original list: ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", originalList[i]);
    }
    printf("\n");
    int userPosition, userNumber;
    printf("Enter the position to insert the number: ");
    scanf("%d", &userPosition);
    printf("Enter the number to insert: ");
    scanf("%d", &userNumber);
    insertNumber(originalList, size, userPosition, userNumber);
    return 0;
}
```

OUTPUT:



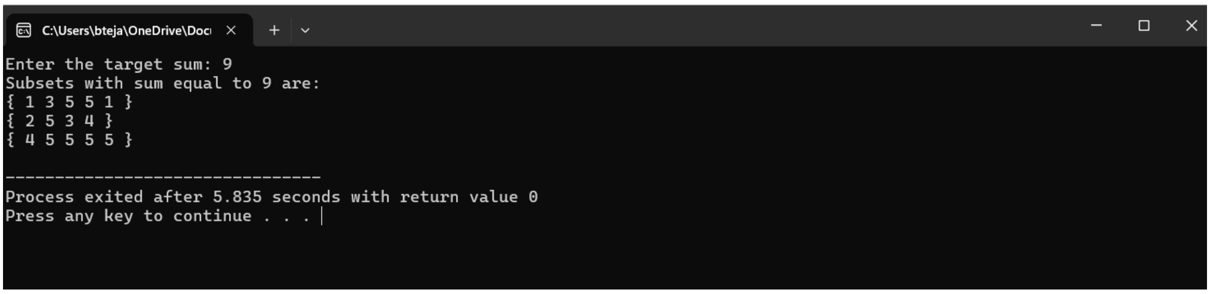
```
Original list: 1 2 3 4 5
Enter the position to insert the number: 3
Enter the number to insert: 9
9 inserted at position 3. Updated list: 1 2 3 9 4 5

--------------------------------
Process exited after 8.839 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

```
#include <stdio.h>

void displaySubset(int subset[], int size) {
    printf("{ ");
    for (int i = 0; i < size; ++i) {
        printf("%d ", subset[i]);
    }
    printf("}\n");
}
void sumOfSubsetsBacktrack(int set[], int subset[], int n, int target, int index, int
currentSum) {
   if (currentSum == target) {
      displaySubset(subset, index);
      return;
   }
   for (int i = index; i < n; ++i) {
      if (currentSum + set[i] <= target) {
         subset[index] = set[i];
         sumOfSubsetsBacktrack(set, subset, n, target, i + 1, currentSum + set[i]);
      }
   }
}
void sumOfSubsets(int set[], int n, int target) {
   int subset[n];
   sumOfSubsetsBacktrack(set, subset, n, target, 0, 0);
}
int main() {
   int inputSet[] = {1, 2, 3, 4, 5};
   int n = sizeof(inputSet) / sizeof(inputSet[0]);
   int targetSum;
   printf("Enter the target sum: ");
   scanf("%d", &targetSum);
   printf("Subsets with sum equal to %d are:\n", targetSum);
   sumOfSubsets(inputSet, n, targetSum);
   return 0;
}
```

OUTPUT:



```
Enter the target sum: 9
Subsets with sum equal to 9 are:
{ 1 3 5 5 1 }
{ 2 5 3 4 }
{ 4 5 5 5 5 }

--------------------------------
Process exited after 5.835 seconds with return value 0
Press any key to continue . . .
```

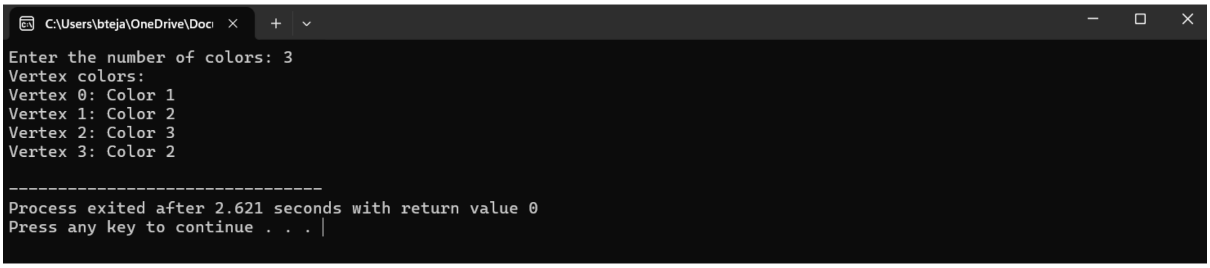
6. Write a program to perform graph coloring problem using backtracking.

PROGRAM:

```
#include <stdio.h>
#define MAX_VERTICES 10
void printSolution(int graphColors[], int n) {
   printf("Vertex colors:\n");
   for (int i = 0; i < n; ++i) {
      printf("Vertex %d: Color %d\n", i, graphColors[i]);
   }
}
```

```
int isSafe(int v, int graphColors[], int graph[MAX_VERTICES][MAX_VERTICES], int
c, int n) {
    for (int i = 0; i < n; ++i) {
        if (graph[v][i] && c == graphColors[i]) {
            return 0;
        }
    }
    return 1;
}
int graphColoringUtil(int v, int graphColors[], int
graph[MAX_VERTICES][MAX_VERTICES], int m, int n) {
    if (v == n) {
        // All vertices are colored
        printSolution(graphColors, n);
        return 1;
    }
    for (int c = 1; c <= m; ++c) {
        if (isSafe(v, graphColors, graph, c, n)) {
            graphColors[v] = c;
            if (graphColoringUtil(v + 1, graphColors, graph, m, n)) {
                return 1;
            }
            graphColors[v] = 0;
        }
    }

    return 0;
}
void graphColoring(int graph[MAX_VERTICES][MAX_VERTICES], int m, int n) {
    int graphColors[MAX_VERTICES];
    for (int i = 0; i < n; ++i) {
        graphColors[i] = 0; // Initialize colors to 0
    }
    if (!graphColoringUtil(0, graphColors, graph, m, n)) {
        printf("No solution exists with %d colors.\n", m);
    }
}
int main() {
    int graphArr[MAX_VERTICES][MAX_VERTICES] = {{0, 1, 1, 1},
                                                {1, 0, 1, 0},
                                                {1, 1, 0, 1},
                                                {1, 0, 1, 0}};
    int numVertices = 4;
    int numColors;
    printf("Enter the number of colors: ");
    scanf("%d", &numColors);
    graphColoring(graphArr, numColors, numVertices);
    return 0;
}
```

OUTPUT:



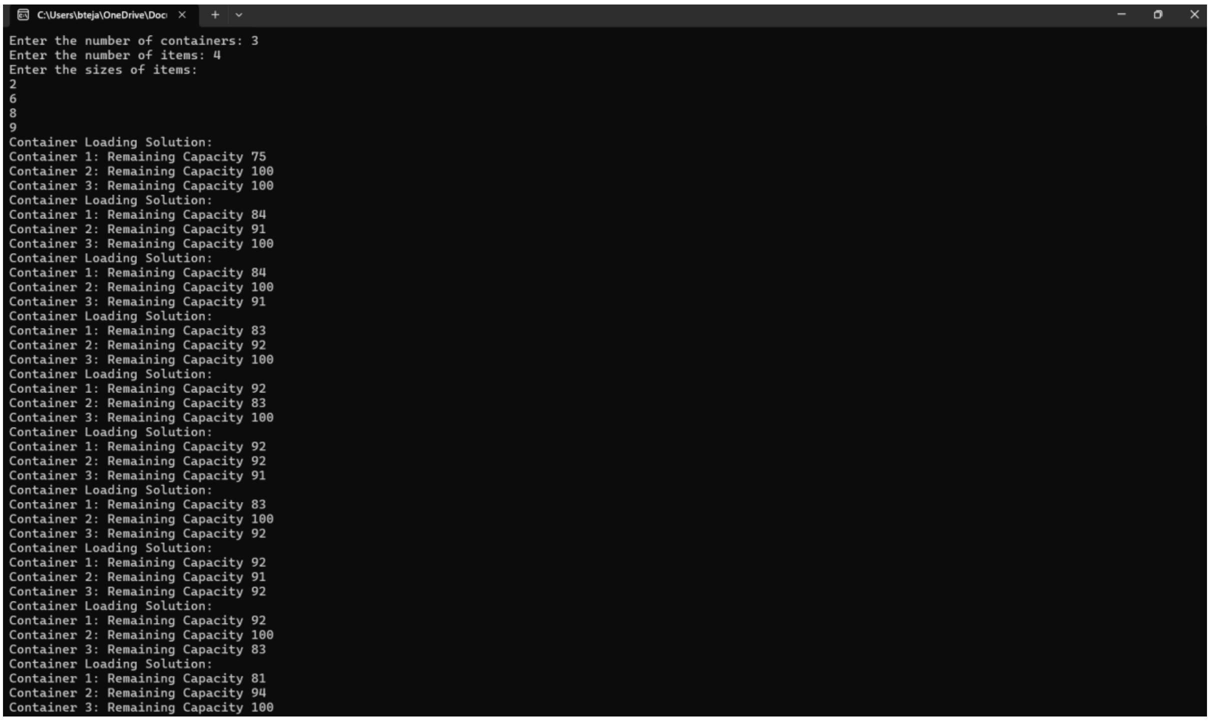
```
Enter the number of colors: 3
Vertex colors:
Vertex 0: Color 1
Vertex 1: Color 2
Vertex 2: Color 3
Vertex 3: Color 2

--------------------------------
Process exited after 2.621 seconds with return value 0
Press any key to continue . . .
```

7. Write a program to compute container loader Problem.

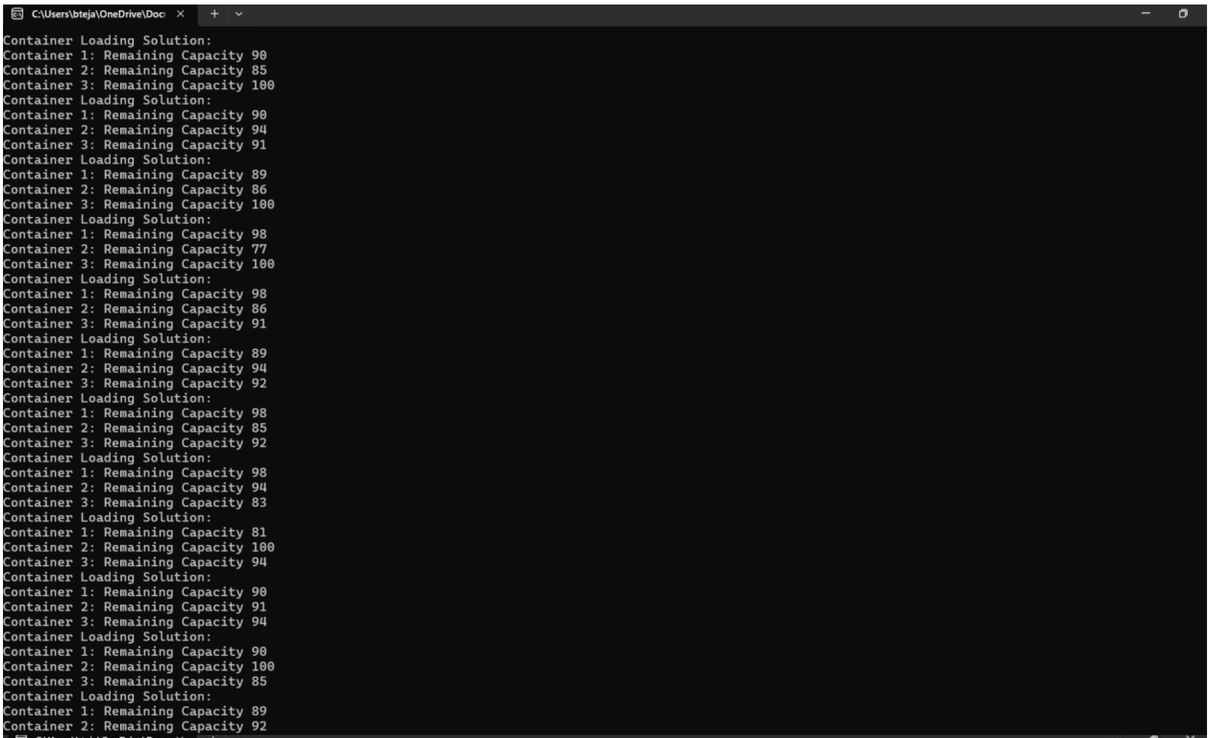
PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ITEMS 20
#define MAX_CONTAINERS 10
int containers[MAX_CONTAINERS];
int items[MAX_ITEMS];
int numContainers;
int numItems;
void printSolution() {
    printf("Container Loading Solution:\n");
    for (int i = 0; i < numContainers; ++i) {
        printf("Container %d: Remaining Capacity %d\n", i + 1, containers[i]);
    }
}
int canPlaceItem(int itemSize, int containerIndex) {
    return containers[containerIndex] >= itemSize;
}
void loadContainers(int itemIndex) {
    if (itemIndex == numItems) {
        printSolution();
        return;
    }
    for (int i = 0; i < numContainers; ++i) {
        if (canPlaceItem(items[itemIndex], i)) {
            containers[i] -= items[itemIndex];
            loadContainers(itemIndex + 1);
            containers[i] += items[itemIndex]; // Backtrack: restore container capacity
        }
    }
}
int main() {
    printf("Enter the number of containers: ");
    scanf("%d", &numContainers);
    printf("Enter the number of items: ");
    scanf("%d", &numItems);
    printf("Enter the sizes of items:\n");
    for (int i = 0; i < numItems; ++i) {
        scanf("%d", &items[i]);
    }
    for (int i = 0; i < numContainers; ++i) {
        containers[i] = 100; // Set initial capacity for each container (assuming 100 in this example)
    }
    loadContainers(0);
    return 0;
}
```

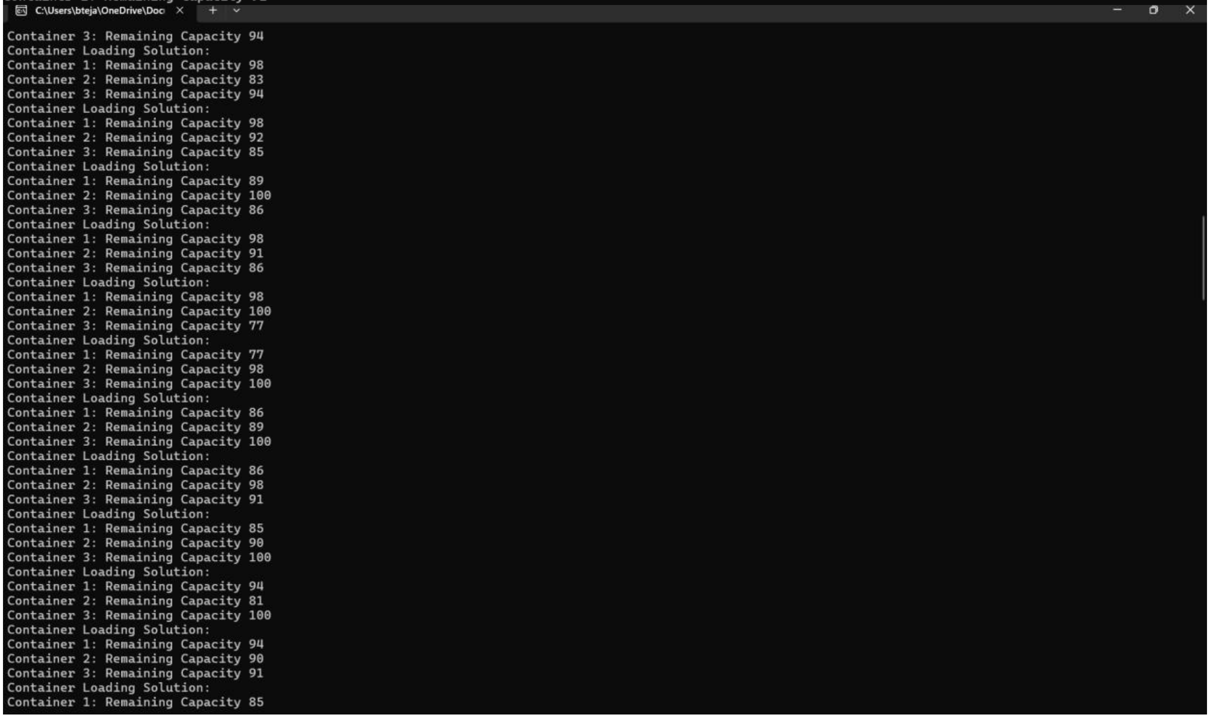
OUTPUT:



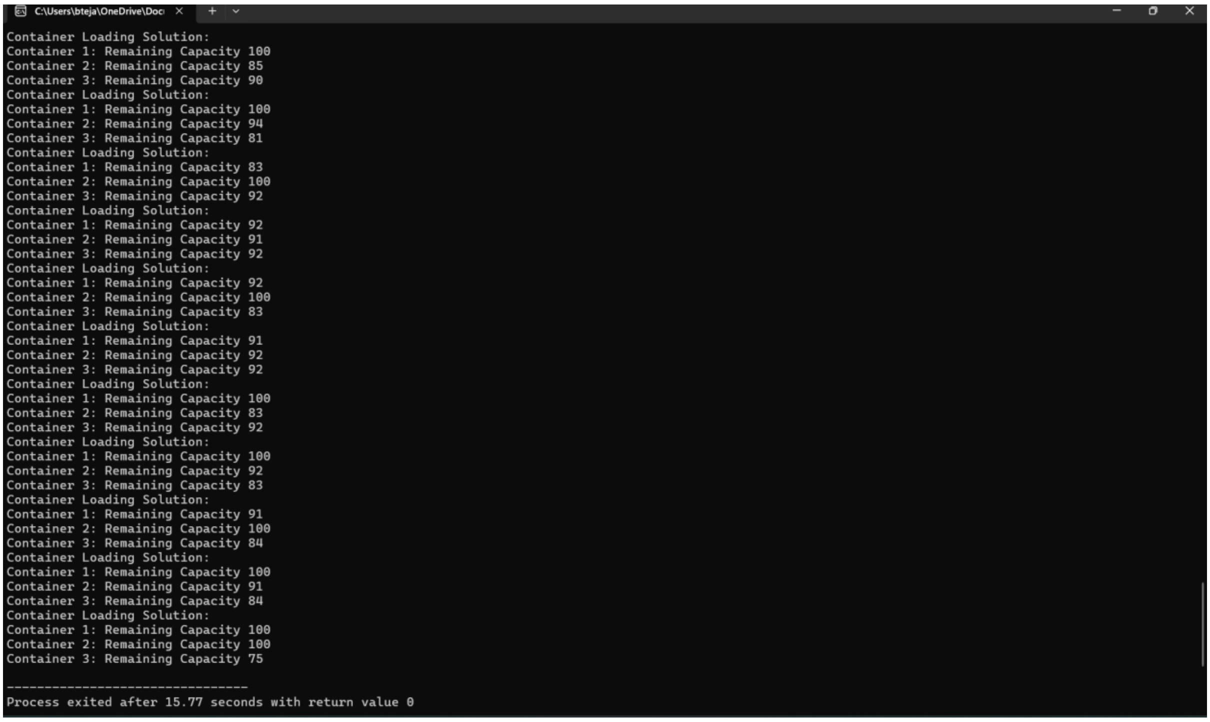
```
Enter the number of containers: 3
Enter the number of items: 4
Enter the sizes of items:
2
6
8
9
Container Loading Solution:
Container 1: Remaining Capacity 75
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 84
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 84
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 83
Container 2: Remaining Capacity 92
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 83
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 92
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 83
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 83
Container Loading Solution:
Container 1: Remaining Capacity 81
Container 2: Remaining Capacity 94
Container 3: Remaining Capacity 100
```



```
Container Loading Solution:
Container 1: Remaining Capacity 90
Container 2: Remaining Capacity 85
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 90
Container 2: Remaining Capacity 94
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 89
Container 2: Remaining Capacity 86
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 77
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 86
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 89
Container 2: Remaining Capacity 94
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 85
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 94
Container 3: Remaining Capacity 83
Container Loading Solution:
Container 1: Remaining Capacity 81
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 94
Container Loading Solution:
Container 1: Remaining Capacity 90
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 94
Container Loading Solution:
Container 1: Remaining Capacity 90
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 85
Container Loading Solution:
Container 1: Remaining Capacity 89
Container 2: Remaining Capacity 92
```



```
Container 3: Remaining Capacity 94
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 83
Container 3: Remaining Capacity 94
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 92
Container 3: Remaining Capacity 85
Container Loading Solution:
Container 1: Remaining Capacity 89
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 86
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 86
Container Loading Solution:
Container 1: Remaining Capacity 98
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 77
Container Loading Solution:
Container 1: Remaining Capacity 77
Container 2: Remaining Capacity 98
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 86
Container 2: Remaining Capacity 89
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 86
Container 2: Remaining Capacity 98
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 85
Container 2: Remaining Capacity 90
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 94
Container 2: Remaining Capacity 81
Container 3: Remaining Capacity 100
Container Loading Solution:
Container 1: Remaining Capacity 94
Container 2: Remaining Capacity 90
Container 3: Remaining Capacity 91
Container Loading Solution:
Container 1: Remaining Capacity 85
```



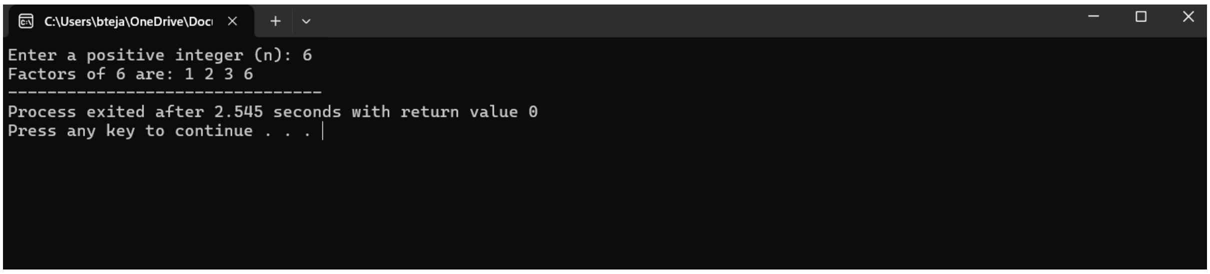
```
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 85
Container 3: Remaining Capacity 90
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 94
Container 3: Remaining Capacity 81
Container Loading Solution:
Container 1: Remaining Capacity 83
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 92
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 83
Container Loading Solution:
Container 1: Remaining Capacity 91
Container 2: Remaining Capacity 92
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 83
Container 3: Remaining Capacity 92
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 92
Container 3: Remaining Capacity 83
Container Loading Solution:
Container 1: Remaining Capacity 91
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 84
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 91
Container 3: Remaining Capacity 84
Container Loading Solution:
Container 1: Remaining Capacity 100
Container 2: Remaining Capacity 100
Container 3: Remaining Capacity 75

--------------------------------
Process exited after 15.77 seconds with return value 0
```

8. Write a program to generate the list of all factor for n value using recursion

PROGRAM:

```
#include <stdio.h>
void generateFactors(int n, int currentFactor) {
    if (currentFactor > n)
        return;
    if (n % currentFactor == 0) {
        printf("%d ", currentFactor);
    }
    generateFactors(n, currentFactor + 1);
}
int main() {
    int n;
    printf("Enter a positive integer (n): ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Please enter a positive integer.\n");
        return 1;
    }
    printf("Factors of %d are: ", n);
    generateFactors(n, 1);
    return 0;
}
```

OUTPUT:



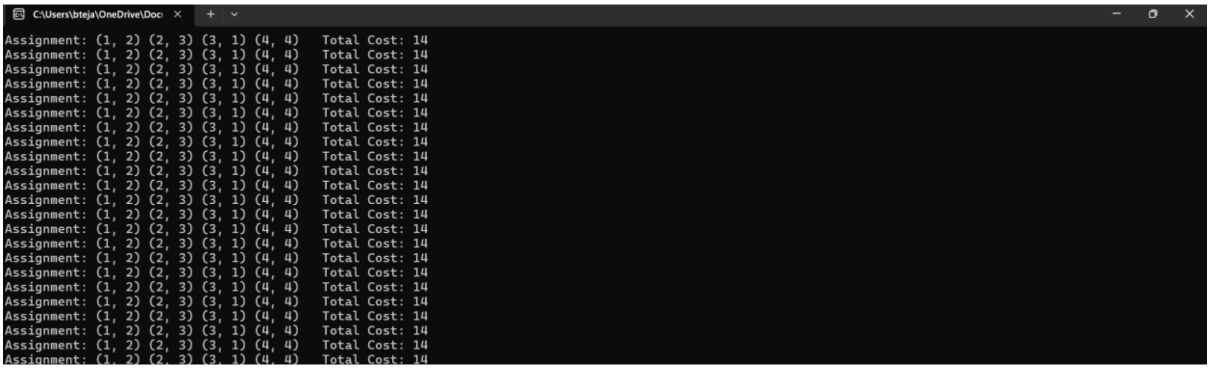
```
Enter a positive integer (n): 6
Factors of 6 are: 1 2 3 6
--------------------------------
Process exited after 2.545 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
#define N 4
int costMatrix[N][N] = {
    {9, 2, 7, 8},
    {6, 4, 3, 7},
    {5, 8, 1, 8},
    {7, 6, 9, 4}
};
int min(int a, int b) {
    return (a < b) ? a : b;
}
void copyArray(int* dest, int* src, int n) {
    for (int i = 0; i < n; ++i) {
        dest[i] = src[i];
    }
}
int reduceRow(int matrix[N][N], int row, int n) {
    int minVal = INT_MAX;
    for (int i = 0; i < n; ++i) {
        if (matrix[row][i] < minVal) {
            minVal = matrix[row][i];
        }
    }
    for (int i = 0; i < n; ++i) {
        matrix[row][i] -= minVal;
    }
    return minVal;
}
int reduceCol(int matrix[N][N], int col, int n) {
    int minVal = INT_MAX;
    for (int i = 0; i < n; ++i) {
        if (matrix[i][col] < minVal) {
            minVal = matrix[i][col];
        }
    }
    for (int i = 0; i < n; ++i) {
        matrix[i][col] -= minVal;
    }
    return minVal;
}
int calculateBound(int matrix[N][N], int n) {
    int bound = 0;
    for (int i = 0; i < n; ++i) {
        bound += reduceRow(matrix, i, n);
        bound += reduceCol(matrix, i, n);
    }
    return bound;
}

void branchAndBound(int matrix[N][N], int assignment[N], int assigned, int totalCost, int
n) {
    if (assigned == n) {
        printf("Assignment: ");
        for (int i = 0; i < n; ++i) {
            printf("(%d, %d) ", i + 1, assignment[i] + 1);
        }
        printf("  Total Cost: %d\n", totalCost);
        return;
    }
    int minCost = INT_MAX;
    int row, col;
    for (int i = 0; i < n; ++i) {
        if (assignment[i] == -1) {
            int tempAssignment[N];
            copyArray(tempAssignment, assignment, n);
            tempAssignment[i] = assigned;

            int tempMatrix[N][N];
            for (int j = 0; j < N; ++j) {
                for (int k = 0; k < N; ++k) {
                    tempMatrix[j][k] = matrix[j][k];
                }
            }
            int cost = matrix[i][assigned] + reduceRow(tempMatrix, i, n) +
reduceCol(tempMatrix, assigned, n);
            int bound = totalCost + cost + calculateBound(tempMatrix, n);
            if (bound < minCost) {
                minCost = bound;
                row = i;
                col = assigned;
            }
        }
    }
    assignment[row] = col;
    int tempMatrix[N][N];
    for (int j = 0; j < N; ++j) {
        for (int k = 0; k < N; ++k) {
            tempMatrix[j][k] = matrix[j][k];
        }
    }
    reduceRow(tempMatrix, row, n);
    reduceCol(tempMatrix, col, n);
    branchAndBound(tempMatrix, assignment, assigned + 1, totalCost + matrix[row][col], 
n);
    assignment[row] = -1;
    branchAndBound(matrix, assignment, assigned, totalCost, n);
}
int main() {
    int assignment[N];
    for (int i = 0; i < N; ++i) {
        assignment[i] = -1;
    }
    printf("Initial Assignment Matrix:\n");
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            printf("%d\t", costMatrix[i][j]);
        }
        printf("\n");
    }
    printf("\nOptimal Assignments and Total Cost:\n");
    branchAndBound(costMatrix, assignment, 0, 0, N);
    return 0;
}
```

OUTPUT:



```
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
Assignment: (1, 2) (2, 3) (3, 1) (4, 4)   Total Cost: 14
```

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

PROGRAM:

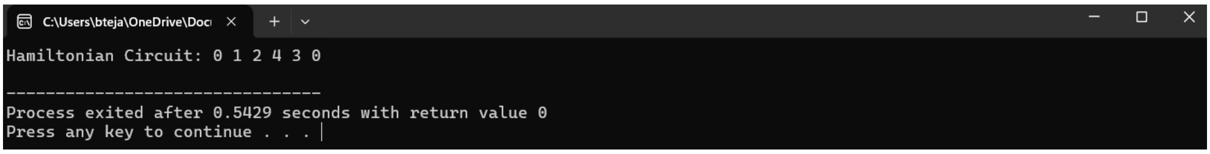
```
#include <stdio.h>

#include <stdbool.h>

#define V 5

void printSolution(int path[V]);
```

```
bool isSafe(int v, bool graph[V][V], int path[V], int pos) {
    if (!graph[path[pos - 1]][v]) {
        return false;
    }
    for (int i = 0; i < pos; ++i) {
        if (path[i] == v) {
            return false;
        }
    }
    return true;
}
bool hamiltonianCircuitUtil(bool graph[V][V], int path[V], int pos) {
    if (pos == V) {
        if (graph[path[pos - 1]][path[0]]) {
            printSolution(path);
            return true;
        } else {
            return false;
        }
    }
    for (int v = 1; v < V; ++v) {
        if (isSafe(v, graph, path, pos)) {
            path[pos] = v;
            if (hamiltonianCircuitUtil(graph, path, pos + 1)) {
                return true;
            }
            path[pos] = -1;
        }
    }
    return false;
```

```
}
bool hamiltonianCircuit(bool graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; ++i) {
        path[i] = -1;
    }
    path[0] = 0;
    if (!hamiltonianCircuitUtil(graph, path, 1)) {
        printf("No Hamiltonian circuit exists.\n");
        return false;
    }
    return true;
}
void printSolution(int path[V]) {
    printf("Hamiltonian Circuit: ");
    for (int i = 0; i < V; ++i) {
        printf("%d ", path[i]);
    }
    printf("%d\n", path[0]);
}
int main() {
    bool graph[V][V] = {
        {0, 1, 1, 1, 0},
        {1, 0, 1, 0, 1},
        {1, 1, 0, 1, 1},
        {1, 0, 1, 0, 1},
        {0, 1, 1, 1, 0}
    };
    hamiltonianCircuit(graph);
    return 0;
```

}

OUTPUT:



```
Hamiltonian Circuit: 0 1 2 4 3 0

--------------------------------
Process exited after 0.5429 seconds with return value 0
Press any key to continue . . .
```