VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Lab Assignment - II, Winter Semester 2023-24

Course Code : BCSE204P	**Slot** : L9 + L10
Course Name: Design and Analysis of Algorithms	**Marks** : 10
Last Date : 05/03/2024	

1. Divide and Conquer Approach

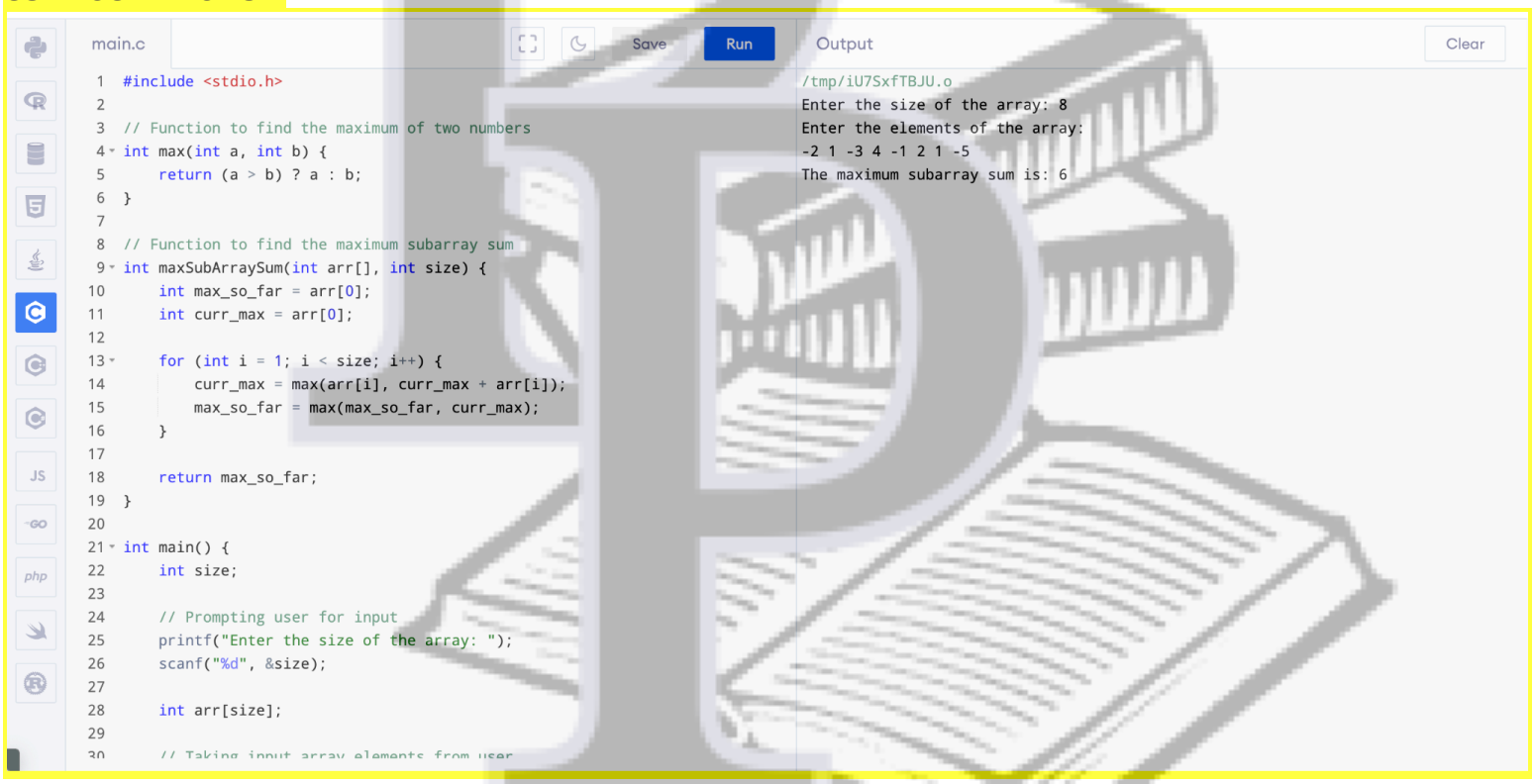
Maximum Subarray Sum

CODE:

```
#include <stdio.h>

// Function to find the maximum of two numbers
int max(int a, int b) {
   return (a > b) ? a : b;
}

// Function to find the maximum subarray sum
int maxSubArraySum(int arr[], int size) {
   int max_so_far = arr[0];
   int curr_max = arr[0];

   for (int i = 1; i < size; i++) {
      curr_max = max(arr[i], curr_max + arr[i]);
      max_so_far = max(max_so_far, curr_max);
   }

   return max_so_far;
}

int main() {
   int size;

   // Prompting user for input
   printf("Enter the size of the array: ");
   scanf("%d", &size);

   int arr[size];
```

```
    // Taking input array elements from user
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    // Finding the maximum subarray sum
    int maxSum = maxSubArraySum(arr, size);

    // Printing the result
    printf("The maximum subarray sum is: %d\n", maxSum);

    return 0;
}
```

CODE SCREENSHOT:



OUTPUT:

Enter the size of the array: 8
Enter the elements of the array:
-2 1 -3 4 -1 2 1 -5
The maximum subarray sum is: 6

OUTPUT SCREENSHOT:

Output Clear

```
/tmp/iU7SxfTBJU.o
Enter the size of the array: 8
Enter the elements of the array:
-2 1 -3 4 -1 2 1 -5
The maximum subarray sum is: 6
```

2. **Backtracking:**

N Queens Problem

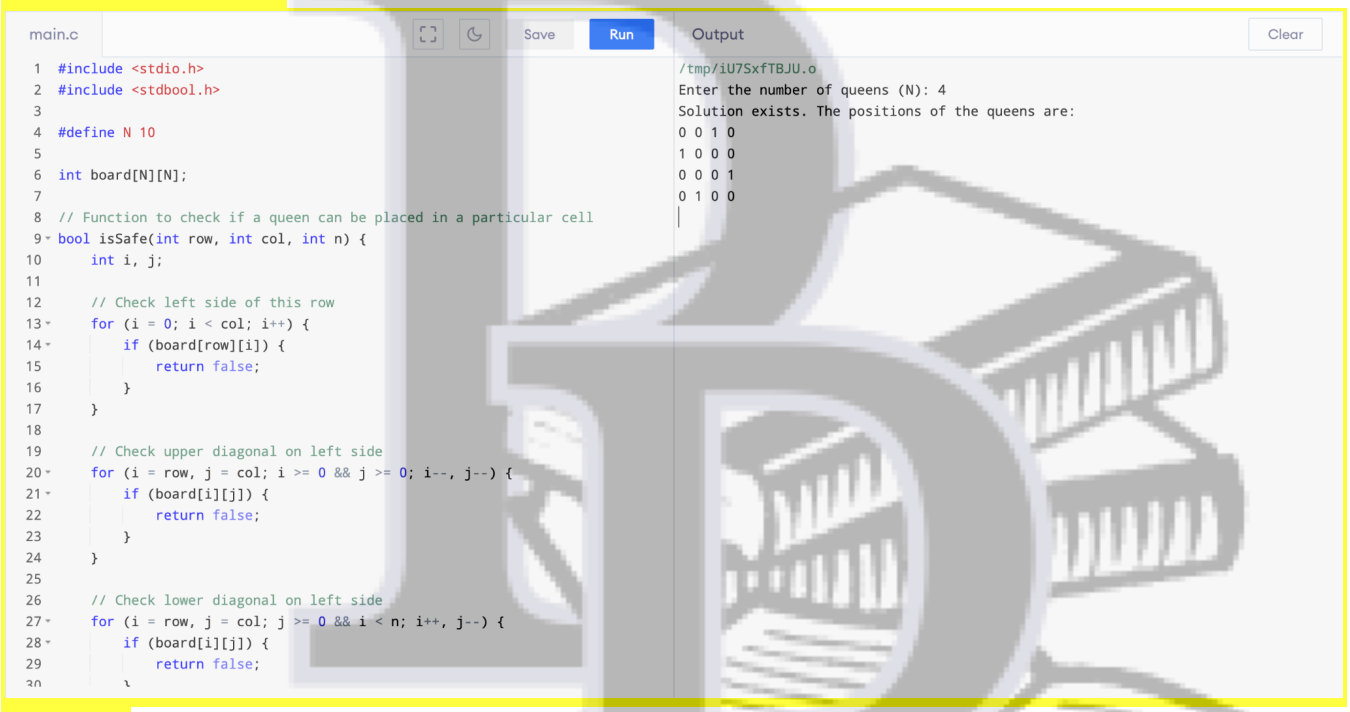
CODE:

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

#define N 10

int board[N][N];

// Function to check if a queen can be placed in a particular cell
bool isSafe(int row, int col, int n) {
   int i, j;

   // Check left side of this row
   for (i = 0; i < col; i++) {
      if (board[row][i]) {
         return false;
      }
   }

   // Check upper diagonal on left side
   for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
      if (board[i][j]) {
         return false;
      }
   }

   // Check lower diagonal on left side
   for (i = row, j = col; j >= 0 && i < n; i++, j--) {
      if (board[i][j]) {
         return false;
      }
   }
```

```
    return true;
}

// Function to solve N Queens problem using backtracking
bool solveNQueens(int col, int n) {
    if (col >= n) {
        return true; // All queens are placed successfully
    }

    for (int i = 0; i < n; i++) {
        if (isSafe(i, col, n)) {
            board[i][col] = 1; // Place the queen

            if (solveNQueens(col + 1, n)) {
                return true;
            }

            // If placing queen in board[i][col] doesn't lead to a solution, then remove the queen from board[i][col]
            board[i][col] = 0;
        }
    }

    // If the queen can't be placed in any row in this column col, then return false
    return false;
}

// Function to print the solution
void printSolution(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", board[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n;

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

    // Initialize the board
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            board[i][j] = 0;
        }
    }
```

```
    if (solveNQueens(0, n)) {
        printf("Solution exists. The positions of the queens are:\n");
        printSolution(n);
    } else {
        printf("Solution does not exist.");
    }

    return 0;
}
```

CODE SCREENSHOT:



OUTPUT:

Enter the number of queens (N): 4
Solution exists. The positions of the queens are:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

OUTPUT SCREENSHOT:

Output Clear

```
/tmp/iU7SxfTBJU.o
Enter the number of queens (N): 4
Solution exists. The positions of the queens are:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

Subset Sum Problem

CODE:

```
#include <stdio.h>

// Function to check if there exists a subset with given sum
int isSubsetSum(int set[], int n, int sum)
{
   // Base Cases
   if (sum == 0)
      return 1;
   if (n == 0)
      return 0;

   // If last element is greater than sum, then ignore it
   if (set[n - 1] > sum)
      return isSubsetSum(set, n - 1, sum);

   // Check if sum can be obtained by including or excluding the last element
   return isSubsetSum(set, n - 1, sum) || isSubsetSum(set, n - 1, sum - set[n - 1]);
}

int main()
{
   int n, sum;
   printf("Enter the number of elements in the set: ");
   scanf("%d", &n);

   int set[n];
   printf("Enter the elements of the set:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &set[i]);
   }
```

```
    printf("Enter the target sum: ");
    scanf("%d", &sum);

    if (isSubsetSum(set, n, sum))
        printf("Subset with sum %d exists.\n", sum);
    else
        printf("No subset with sum %d exists.\n", sum);

    return 0;
}
```

CODE SCREENSHOT:

OUTPUT:

Enter the number of elements in the set: 5
Enter the elements of the set:
2 3 7 8 10
Enter the target sum: 11
Subset with sum 11 exists.

OUTPUT SCREENSHOT:

```
Output                                                    Clear
/tmp/iU7SxfTBJU.o
Enter the number of elements in the set: 5
Enter the elements of the set:
2 3 7 8 10
Enter the target sum: 11
Subset with sum 11 exists.
```

3. **Dynamic Programming:**

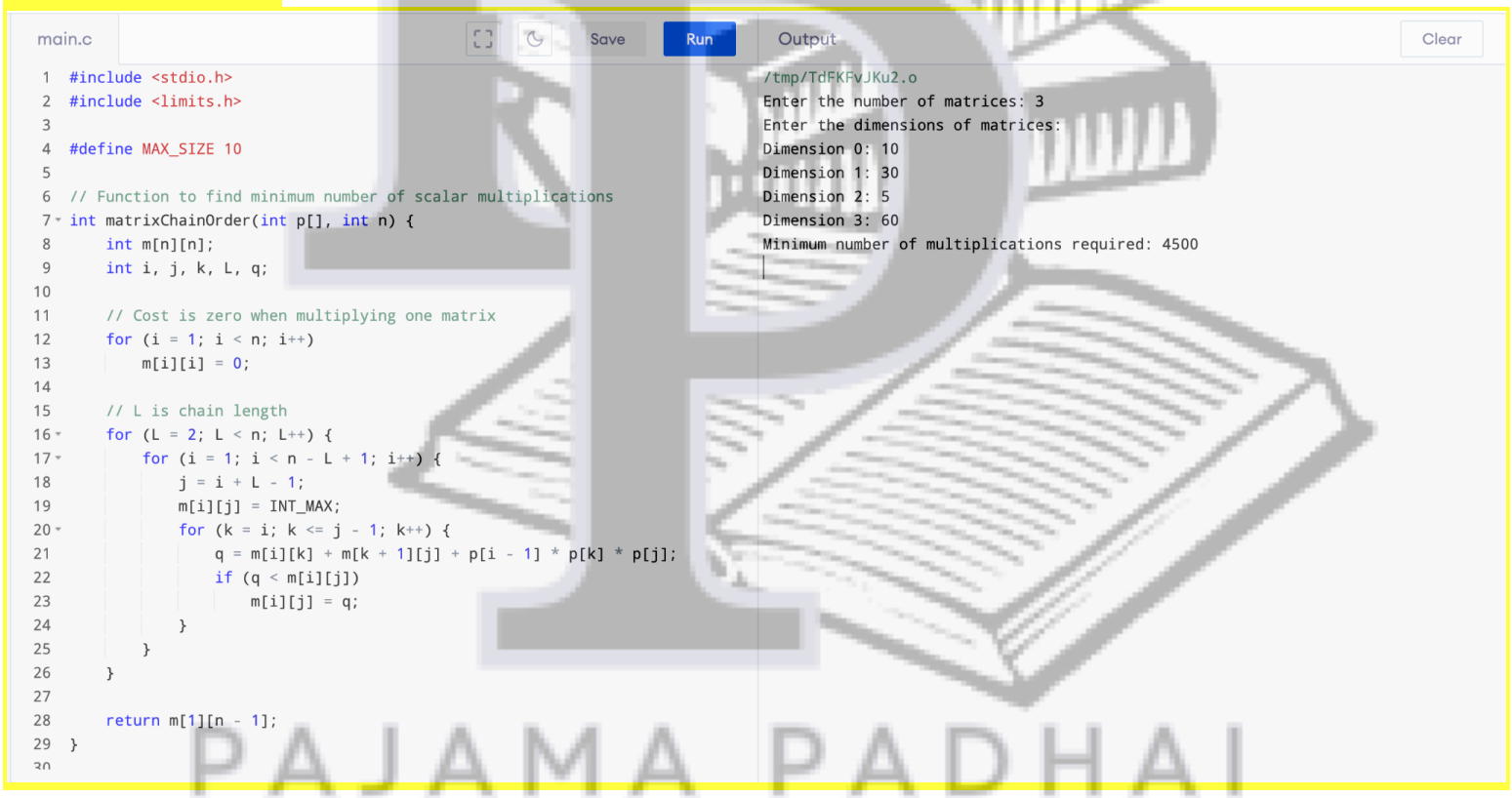
Matrix Chain Multiplication

CODE:

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

#define MAX_SIZE 10

// Function to find minimum number of scalar multiplications
int matrixChainOrder(int p[], int n) {
    int m[n][n];
    int i, j, k, L, q;

    // Cost is zero when multiplying one matrix
    for (i = 1; i < n; i++)
        m[i][i] = 0;

    // L is chain length
    for (L = 2; L < n; L++) {
        for (i = 1; i < n - L + 1; i++) {
            j = i + L - 1;
            m[i][j] = INT_MAX;
            for (k = i; k <= j - 1; k++) {
                q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n - 1];
}
```

```
int main() {
   int n, i;
   int p[MAX_SIZE]; // Array to store dimensions of matrices

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

   printf("Enter the dimensions of matrices:\n");
   for (i = 0; i <= n; i++) {
      printf("Dimension %d: ", i);
      scanf("%d", &p[i]);
   }

   int minMultiplications = matrixChainOrder(p, n + 1);
   printf("Minimum number of multiplications required: %d\n", minMultiplications);

   return 0;
}
```

CODE SCREENSHOT:



OUTPUT:

```
Enter the number of matrices: 3
Enter the dimensions of matrices:
Dimension 0: 10
Dimension 1: 30
Dimension 2: 5
```

Dimension 3: 60
Minimum number of multiplications required: 4500

OUTPUT SCREENSHOT:

Output Clear

```
/tmp/TdFKFvJKu2.o
Enter the number of matrices: 3
Enter the dimensions of matrices:
Dimension 0: 10
Dimension 1: 30
Dimension 2: 5
Dimension 3: 60
Minimum number of multiplications required: 4500
```

Longest Common Subsequence

CODE:

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

#define MAX_LENGTH 100

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

int lcs_length(char *X, char *Y, int m, int n) {
   int L[m + 1][n + 1];
   int i, j;

   for (i = 0; i <= m; i++) {
      for (j = 0; j <= n; j++) {
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i - 1] == Y[j - 1])
            L[i][j] = L[i - 1][j - 1] + 1;
         else
            L[i][j] = max(L[i - 1][j], L[i][j - 1]);
      }
   }

   return L[m][n];
```

```
}

void print_lcs(char *X, char *Y, int m, int n) {
    int L[m + 1][n + 1];
    int index = lcs_length(X, Y, m, n);
    char lcs[index + 1];
    lcs[index] = '\0';

    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs[index - 1] = X[i - 1];
            i--;
            j--;
            index--;
        } else if (L[i - 1][j] > L[i][j - 1])
            i--;
        else
            j--;
    }

    printf("Longest Common Subsequence: %s\n", lcs);
}

int main() {
    char X[MAX_LENGTH], Y[MAX_LENGTH];

    printf("Enter first string: ");
    fgets(X, MAX_LENGTH, stdin);
    X[strcspn(X, "\n")] = '\0'; // Remove trailing newline

    printf("Enter second string: ");
    fgets(Y, MAX_LENGTH, stdin);
    Y[strcspn(Y, "\n")] = '\0'; // Remove trailing newline

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of Longest Common Subsequence: %d\n", lcs_length(X, Y, m, n));
    print_lcs(X, Y, m, n);

    return 0;
}
```


CODE SCREENSHOT:

main.c Save Run Output Clear

```c
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 100

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

int lcs_length(char *X, char *Y, int m, int n) {
    int L[m + 1][n + 1];
    int i, j;

    for (i = 0; i <= m; i++) {
        for (j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    return L[m][n];
}

void print_lcs(char *X, char *Y, int m, int n) {
    int L[m + 1][n + 1];
    int index = lcs_length(X, Y, m, n);
```

```
/tmp/TdFKFvJKu2.o
Enter first string: abcd
Enter second string: ab
Length of Longest Common Subsequence: 2
Longest Common Subsequence: ab
```

OUTPUT:

Enter first string: abcd
Enter second string: ab
Length of Longest Common Subsequence: 2
Longest Common Subsequence: ab

OUTPUT SCREENSHOT:

Output Clear

```
/tmp/TdFKFvJKu2.o
Enter first string: abcd
Enter second string: ab
Length of Longest Common Subsequence: 2
Longest Common Subsequence: ab
```


Assembly Line Scheduling

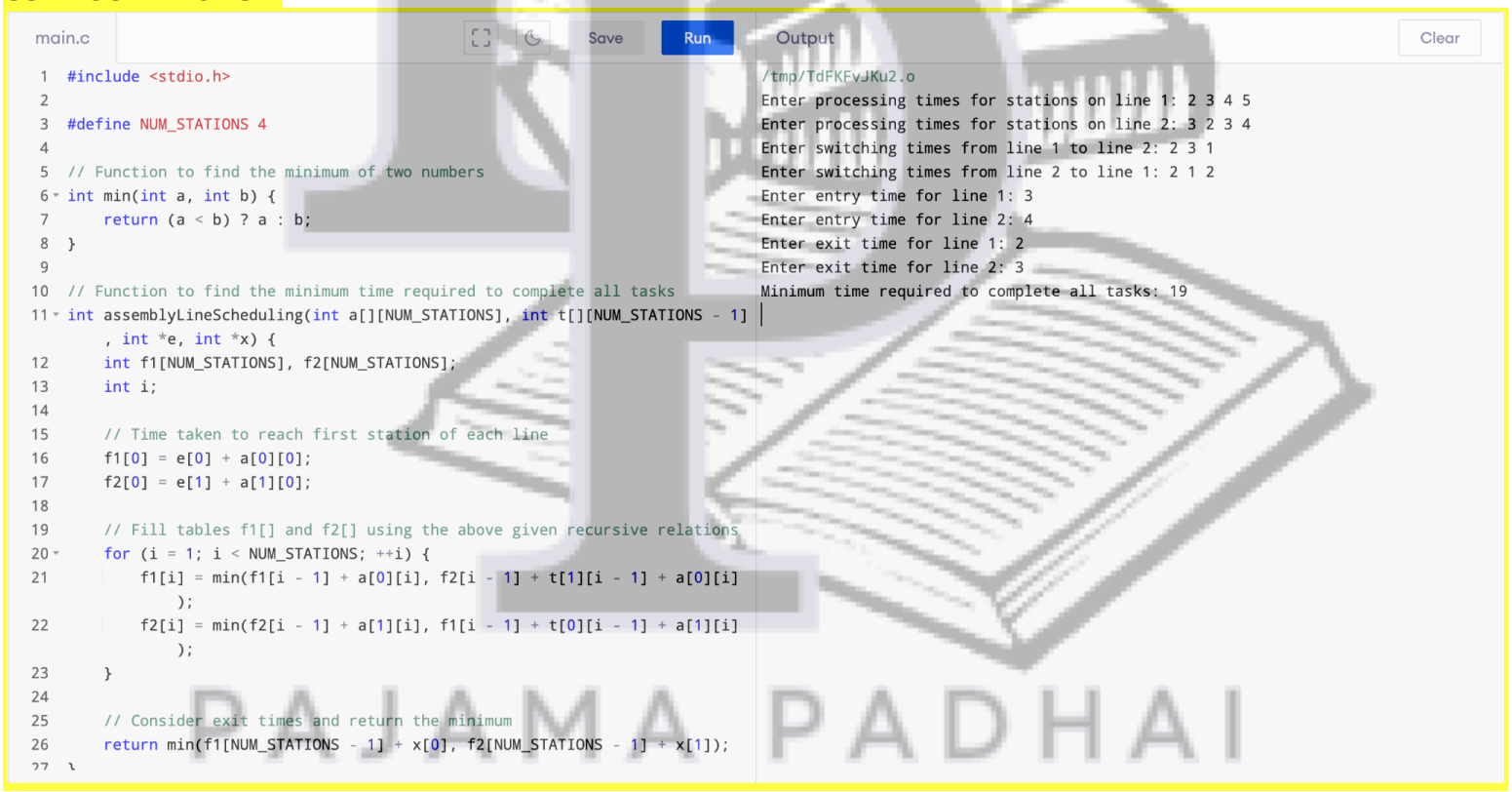
CODE:

```
#include <stdio.h>

#define NUM_STATIONS 4
```

```
// Function to find the minimum of two numbers
int min(int a, int b) {
   return (a < b) ? a : b;
}

// Function to find the minimum time required to complete all tasks
int assemblyLineScheduling(int a[][NUM_STATIONS], int t[][NUM_STATIONS - 1], int *e, int *x) {
   int f1[NUM_STATIONS], f2[NUM_STATIONS];
   int i;

   // Time taken to reach first station of each line
   f1[0] = e[0] + a[0][0];
   f2[0] = e[1] + a[1][0];

   // Fill tables f1[] and f2[] using the above given recursive relations
   for (i = 1; i < NUM_STATIONS; ++i) {
      f1[i] = min(f1[i - 1] + a[0][i], f2[i - 1] + t[1][i - 1] + a[0][i]);
      f2[i] = min(f2[i - 1] + a[1][i], f1[i - 1] + t[0][i - 1] + a[1][i]);
   }

   // Consider exit times and return the minimum
   return min(f1[NUM_STATIONS - 1] + x[0], f2[NUM_STATIONS - 1] + x[1]);
}

int main() {
   int a[2][NUM_STATIONS];  // Time taken at each station on both lines
   int t[2][NUM_STATIONS - 1];  // Time taken to switch lines between stations
   int e[2], x[2];  // Entry and exit times for both lines
   int i, j;

   // Input processing times for each station on both lines
   printf("Enter processing times for stations on line 1: ");
   for (i = 0; i < NUM_STATIONS; ++i) {
      scanf("%d", &a[0][i]);
   }

   printf("Enter processing times for stations on line 2: ");
   for (i = 0; i < NUM_STATIONS; ++i) {
      scanf("%d", &a[1][i]);
   }

   // Input time taken to switch lines between stations
   printf("Enter switching times from line 1 to line 2: ");
   for (i = 0; i < NUM_STATIONS - 1; ++i) {
      scanf("%d", &t[0][i]);
   }

   printf("Enter switching times from line 2 to line 1: ");
   for (i = 0; i < NUM_STATIONS - 1; ++i) {
      scanf("%d", &t[1][i]);
```

```
    }

    // Input entry and exit times for both lines
    printf("Enter entry time for line 1: ");
    scanf("%d", &e[0]);

    printf("Enter entry time for line 2: ");
    scanf("%d", &e[1]);

    printf("Enter exit time for line 1: ");
    scanf("%d", &x[0]);

    printf("Enter exit time for line 2: ");
    scanf("%d", &x[1]);

    // Calculate and print the minimum time required to complete all tasks
    printf("Minimum time required to complete all tasks: %d\n", assemblyLineScheduling(a, t, e, x));

    return 0;
}
```

CODE SCREENSHOT:

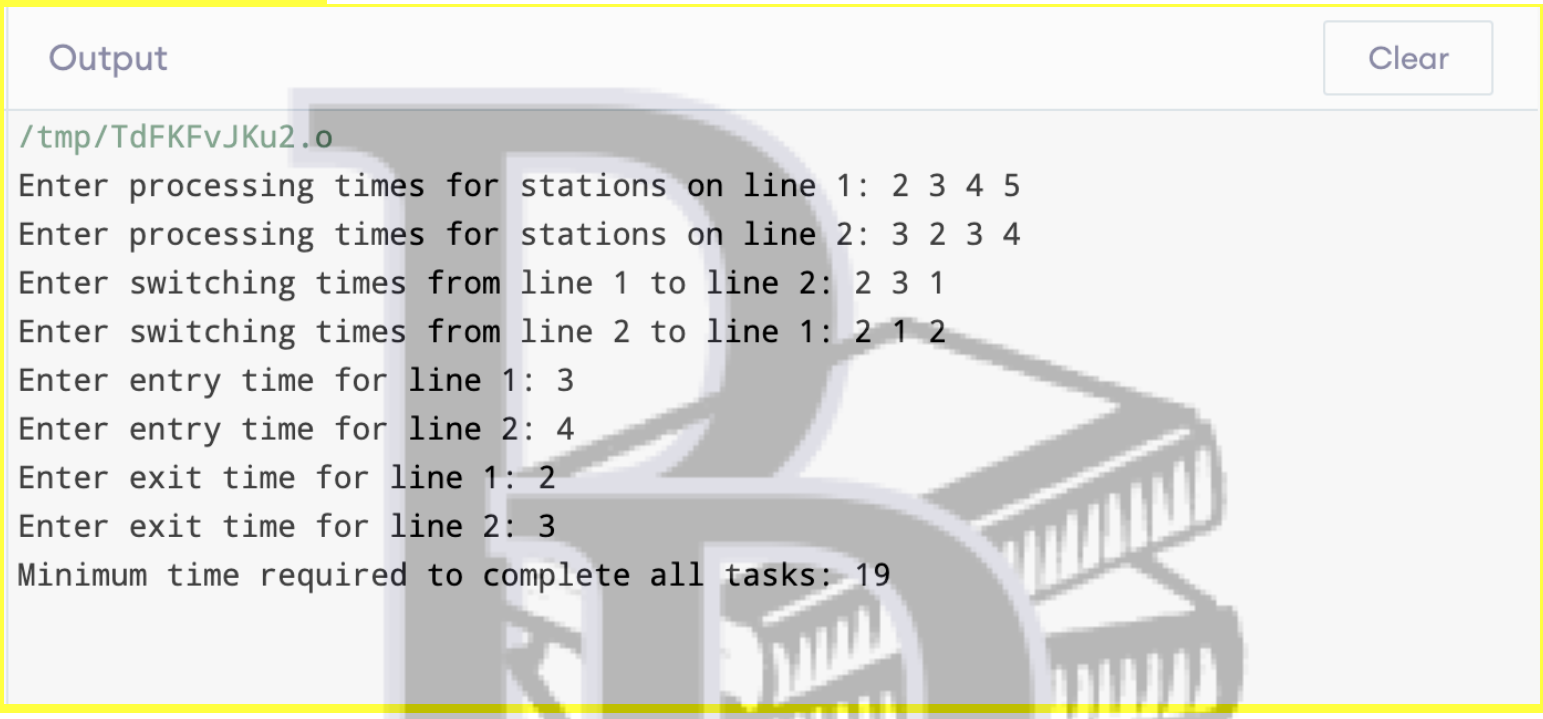


OUTPUT:

Enter processing times for stations on line 1: 2 3 4 5
Enter processing times for stations on line 2: 3 2 3 4
Enter switching times from line 1 to line 2: 2 3 1
Enter switching times from line 2 to line 1: 2 1 2

Enter entry time for line 1: 3
Enter entry time for line 2: 4
Enter exit time for line 1: 2
Enter exit time for line 2: 3
Minimum time required to complete all tasks: 19

OUTPUT SCREENSHOT:



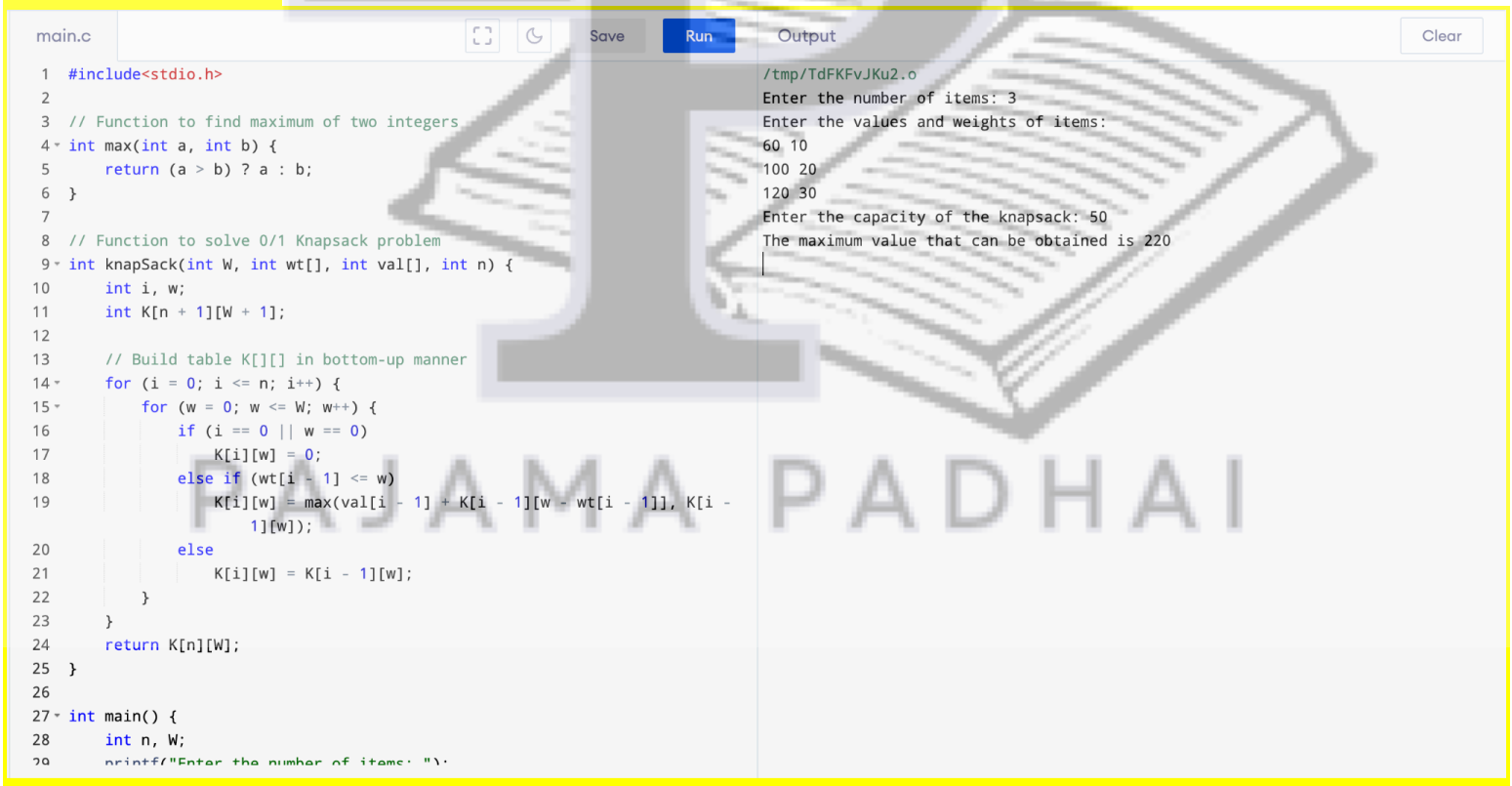
0/1 Knapsack Problem

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 0/1 Knapsack problem
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 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 obtained is %d\n", knapSack(W, wt, val, n));

    return 0;
}
```

CODE SCREENSHOT:



OUTPUT:

Enter the number of items: 3
Enter the values and weights of items:
60 10
100 20
120 30
Enter the capacity of the knapsack: 50
The maximum value that can be obtained is 220

OUTPUT SCREENSHOT: