



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]);
}

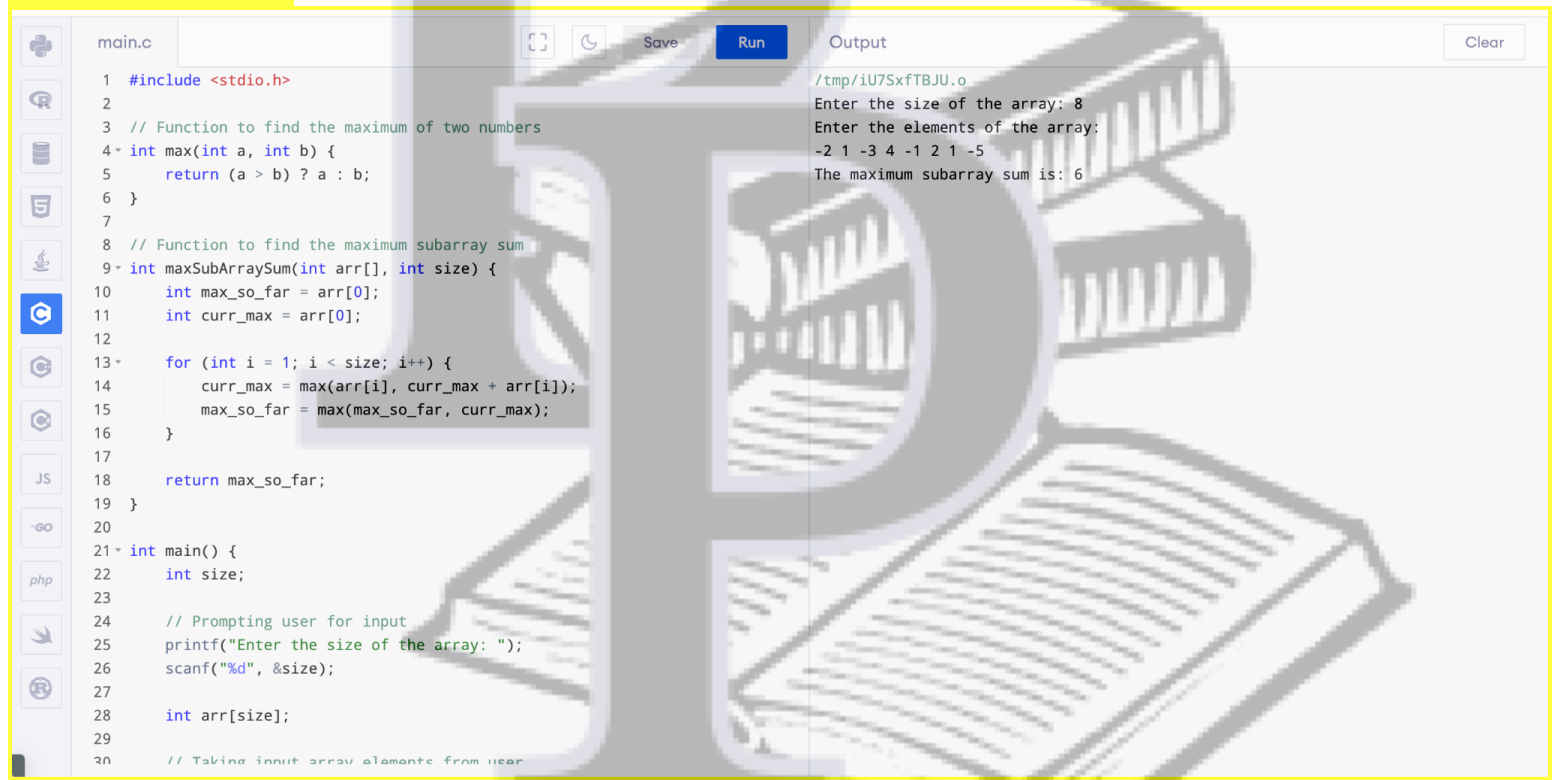
// Finding the maximum subarray sum
int maxSum = maxSubArraySum(arr, size);

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

return 0;
}

```

CODE SCREENSHOT:



The screenshot shows a C++ IDE with a file named 'main.c'. The code implements a function to find the maximum subarray sum using Kadane's algorithm. The main function prompts the user for the array size and elements. The output window shows the execution results.

```

main.c
1 #include <stdio.h>
2
3 // Function to find the maximum of two numbers
4 int max(int a, int b) {
5     return (a > b) ? a : b;
6 }
7
8 // Function to find the maximum subarray sum
9 int maxSubArraySum(int arr[], int size) {
10     int max_so_far = arr[0];
11     int curr_max = arr[0];
12
13     for (int i = 1; i < size; i++) {
14         curr_max = max(arr[i], curr_max + arr[i]);
15         max_so_far = max(max_so_far, curr_max);
16     }
17
18     return max_so_far;
19 }
20
21 int main() {
22     int size;
23
24     // Prompting user for input
25     printf("Enter the size of the array: ");
26     scanf("%d", &size);
27
28     int arr[size];
29
30     // Taking input array elements from user

```

Output

```

/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

```

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;
        }
    }
}

```

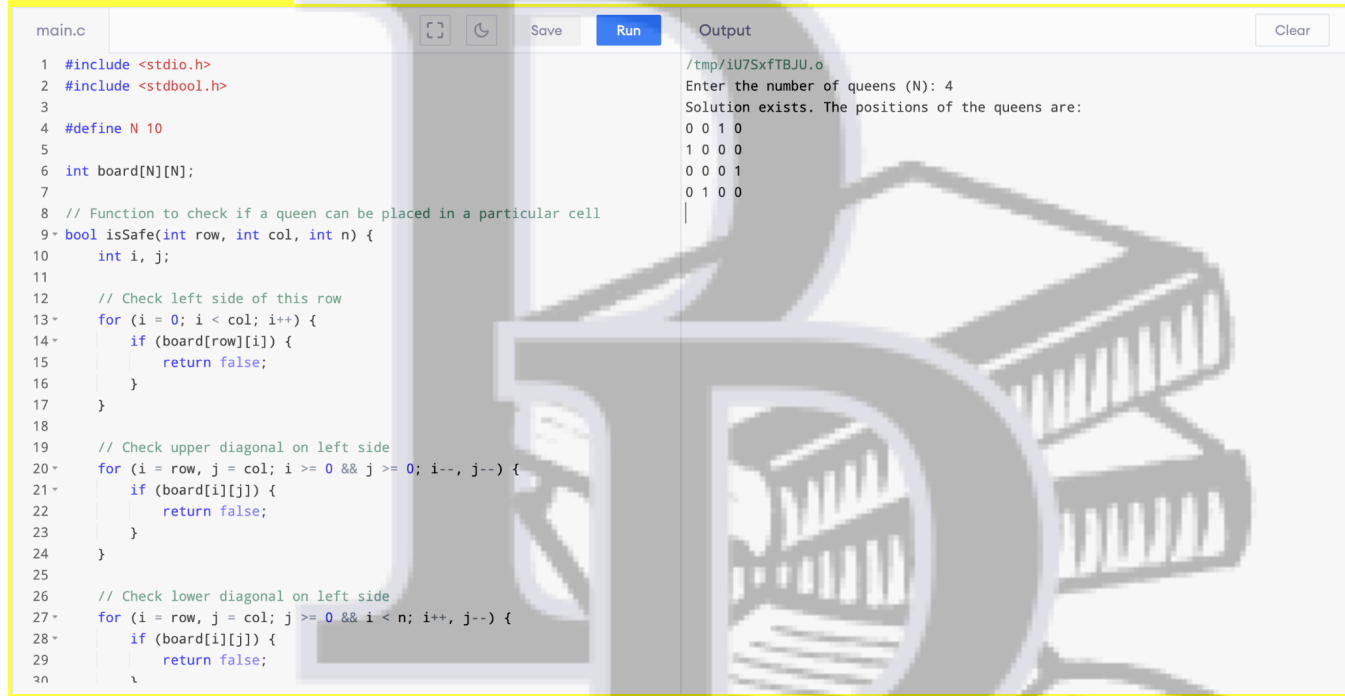
```

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:



```

main.c
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define N 10
5
6 int board[N][N];
7
8 // Function to check if a queen can be placed in a particular cell
9 bool isSafe(int row, int col, int n) {
10     int i, j;
11
12     // Check left side of this row
13     for (i = 0; i < col; i++) {
14         if (board[row][i]) {
15             return false;
16         }
17     }
18
19     // Check upper diagonal on left side
20     for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
21         if (board[i][j]) {
22             return false;
23         }
24     }
25
26     // Check lower diagonal on left side
27     for (i = row, j = col; j >= 0 && i < n; i++, j--) {
28         if (board[i][j]) {
29             return false;
30         }
31     }
32     return true;
33 }
34
35 void printSolution(int n) {
36     for (int i = 0; i < n; i++) {
37         for (int j = 0; j < n; j++) {
38             printf("%d ", board[i][j]);
39         }
40         printf("\n");
41     }
42 }
43
44 int solveNQueens(int row, int n) {
45     if (row == n) {
46         printSolution(n);
47         return true;
48     }
49     for (int col = 0; col < n; col++) {
50         if (isSafe(row, col, n)) {
51             board[row][col] = 1;
52             if (solveNQueens(row + 1, n)) {
53                 return true;
54             }
55             board[row][col] = 0;
56         }
57     }
58     return false;
59 }
60
61 int main() {
62     int n;
63     printf("Enter the number of queens (N): ");
64     scanf("%d", &n);
65     printf("Solution exists. The positions of the queens are:\n");
66     if (solveNQueens(0, n)) {
67         printSolution(n);
68     } else {
69         printf("Solution does not exist.");
70     }
71     return 0;
72 }

```

Output

```

/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

```

OUTPUT:

```

Enter the number of queens (N): 4
Solution exists. The positions of the queens are:
0010
1000
0001
0100

```

OUTPUT SCREENSHOT:

PAJAMA PADHAI

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]);
```

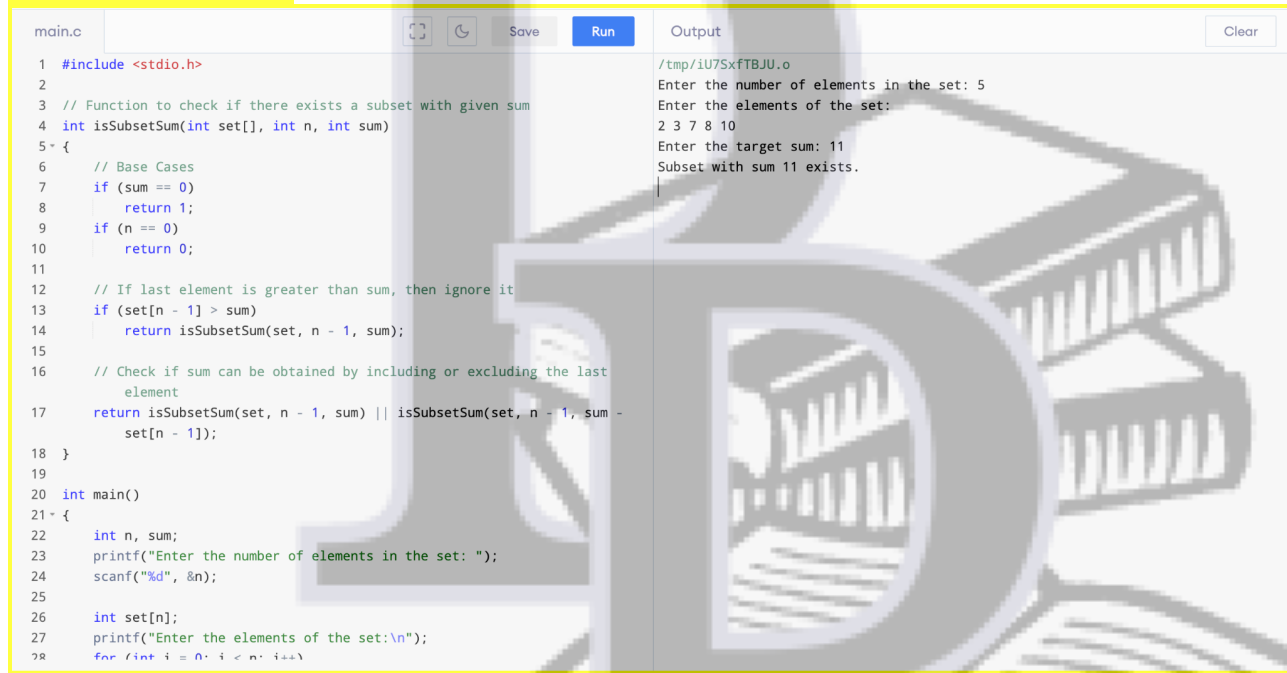
```
    }
```

```
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:



The screenshot shows a C++ IDE with a file named 'main.cpp'. The code implements a recursive function 'isSubsetSum' to check if a subset of a given set of numbers can sum up to a target value. The main function prompts the user for the number of elements, the elements themselves, and the target sum. The output window shows the program's execution: it asks for the number of elements (5), the elements (2 3 7 8 10), the target sum (11), and confirms that a subset with the sum 11 exists.

```
main.cpp
1 #include <stdio.h>
2
3 // Function to check if there exists a subset with given sum
4 int isSubsetSum(int set[], int n, int sum)
5 {
6     // Base Cases
7     if (sum == 0)
8         return 1;
9     if (n == 0)
10        return 0;
11
12    // If last element is greater than sum, then ignore it
13    if (set[n - 1] > sum)
14        return isSubsetSum(set, n - 1, sum);
15
16    // Check if sum can be obtained by including or excluding the last
17    // element
18    return isSubsetSum(set, n - 1, sum) || isSubsetSum(set, n - 1, sum -
19        set[n - 1]);
20 }
21
22 int main()
23 {
24     int n, sum;
25     printf("Enter the number of elements in the set: ");
26     scanf("%d", &n);
27
28     int set[n];
29     printf("Enter the elements of the set:\n");
30     for (int i = 0; i < n; i++)
```

Output

```
/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.
```

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:

PAJAMA PADHAI

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:

The screenshot shows a C++ IDE with a code editor on the left and an output window on the right. The code implements the matrix chain order algorithm. The output window shows the program's execution, including prompts for the number of matrices and their dimensions, followed by the calculated minimum number of multiplications required.

```

main.c
1  #include <stdio.h>
2  #include <limits.h>
3
4  #define MAX_SIZE 10
5
6  // Function to find minimum number of scalar multiplications
7  int matrixChainOrder(int p[], int n) {
8      int m[n][n];
9      int i, j, k, L, q;
10
11     // Cost is zero when multiplying one matrix
12     for (i = 1; i < n; i++)
13         m[i][i] = 0;
14
15     // L is chain length
16     for (L = 2; L < n; L++) {
17         for (i = 1; i < n - L + 1; i++) {
18             j = i + L - 1;
19             m[i][j] = INT_MAX;
20             for (k = i; k <= j - 1; k++) {
21                 q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
22                 if (q < m[i][j])
23                     m[i][j] = q;
24             }
25         }
26     }
27
28     return m[1][n - 1];
29 }
30

```

Output:

```

/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

```

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

SaveRun

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_LENGTH 100
5
6 int max(int a, int b) {
7     return (a > b) ? a : b;
8 }
9
10 int lcs_length(char *X, char *Y, int m, int n) {
11     int L[m + 1][n + 1];
12     int i, j;
13
14     for (i = 0; i <= m; i++) {
15         for (j = 0; j <= n; j++) {
16             if (i == 0 || j == 0)
17                 L[i][j] = 0;
18             else if (X[i - 1] == Y[j - 1])
19                 L[i][j] = L[i - 1][j - 1] + 1;
20             else
21                 L[i][j] = max(L[i - 1][j], L[i][j - 1]);
22         }
23     }
24
25     return L[m][n];
26 }
27
28 void print_lcs(char *X, char *Y, int m, int n) {
29     int L[m + 1][n + 1];
30     int index = lcs_length(X, Y, m, n);
```

Output

Clear

```
/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:

```

main.c
1 #include <stdio.h>
2
3 #define NUM_STATIONS 4
4
5 // Function to find the minimum of two numbers
6 int min(int a, int b) {
7     return (a < b) ? a : b;
8 }
9
10 // Function to find the minimum time required to complete all tasks
11 int assemblyLineScheduling(int a[][NUM_STATIONS], int t[][NUM_STATIONS - 1]
    , int *e, int *x) {
12     int f1[NUM_STATIONS], f2[NUM_STATIONS];
13     int i;
14
15     // Time taken to reach first station of each line
16     f1[0] = e[0] + a[0][0];
17     f2[0] = e[1] + a[1][0];
18
19     // Fill tables f1[] and f2[] using the above given recursive relations
20     for (i = 1; i < NUM_STATIONS; ++i) {
21         f1[i] = min(f1[i - 1] + a[0][i], f2[i - 1] + t[1][i - 1] + a[0][i]
            );
22         f2[i] = min(f2[i - 1] + a[1][i], f1[i - 1] + t[0][i - 1] + a[1][i]
            );
23     }
24
25     // Consider exit times and return the minimum
26     return min(f1[NUM_STATIONS - 1] + x[0], f2[NUM_STATIONS - 1] + x[1]);
27 }

```

Output

```

/tmp/TdFKFvJKu2.o
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:

```

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:

Output

Clear

```
/tmp/TdFKFvJKu2.o
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
```

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]);
    }

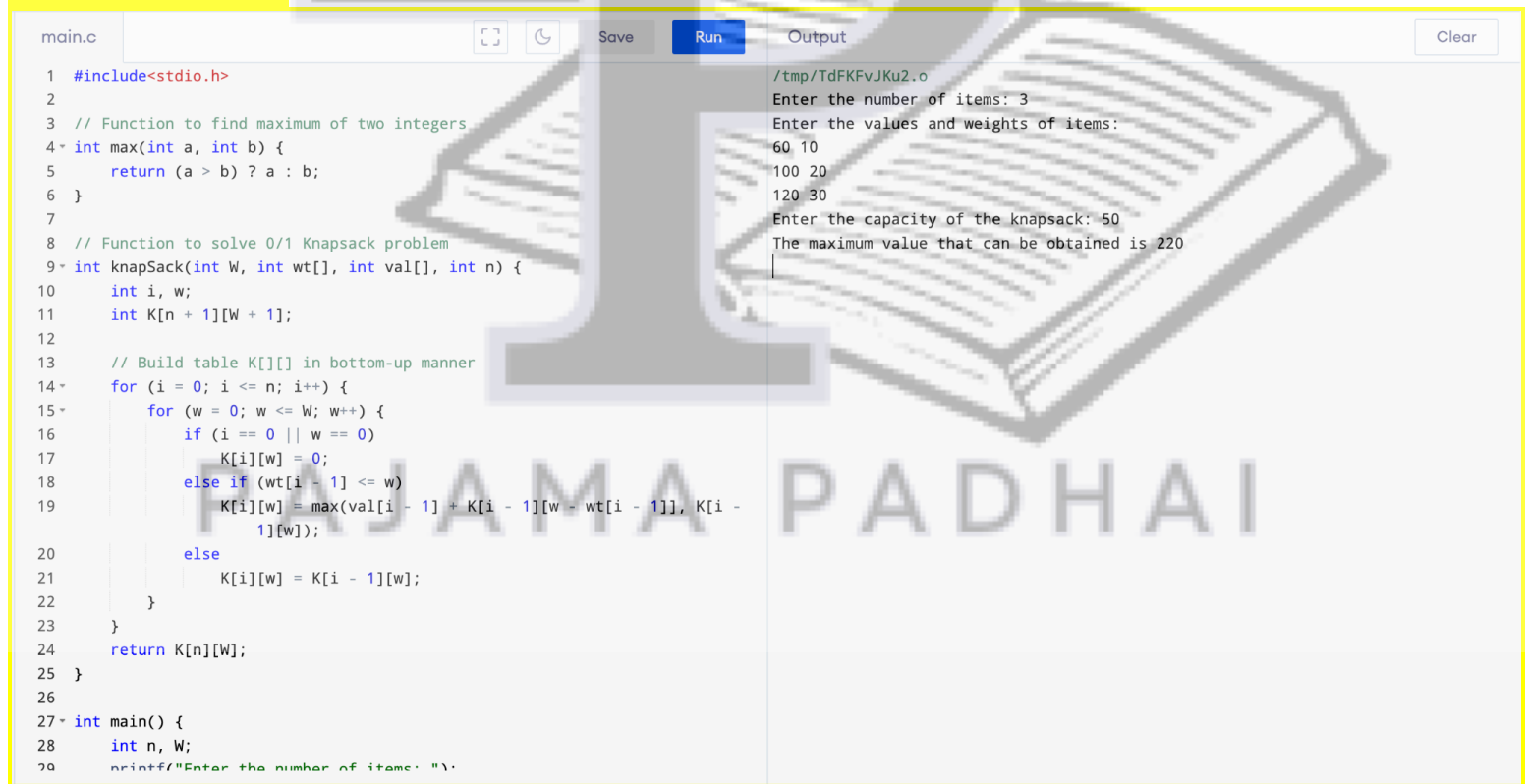
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &W);

    printf("The maximum value that can be obtained is %d\n", knapSack(W, wt, val, n));

    return 0;
}

```

CODE SCREENSHOT:



```

main.c
1 #include<stdio.h>
2
3 // Function to find maximum of two integers
4 int max(int a, int b) {
5     return (a > b) ? a : b;
6 }
7
8 // Function to solve 0/1 Knapsack problem
9 int knapSack(int W, int wt[], int val[], int n) {
10     int i, w;
11     int K[n + 1][W + 1];
12
13     // Build table K[][] in bottom-up manner
14     for (i = 0; i <= n; i++) {
15         for (w = 0; w <= W; w++) {
16             if (i == 0 || w == 0)
17                 K[i][w] = 0;
18             else if (wt[i - 1] <= w)
19                 K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
20             else
21                 K[i][w] = K[i - 1][w];
22         }
23     }
24     return K[n][W];
25 }
26
27 int main() {
28     int n, W;
29     printf("Enter the number of items: ");

```

```

/tmp/TdFKFvJKu2.o
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:

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:

Output

Clear

/tmp/TdFKFvJKu2.o

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

PAJAMA PADHAI