# Analysis of Algorithm

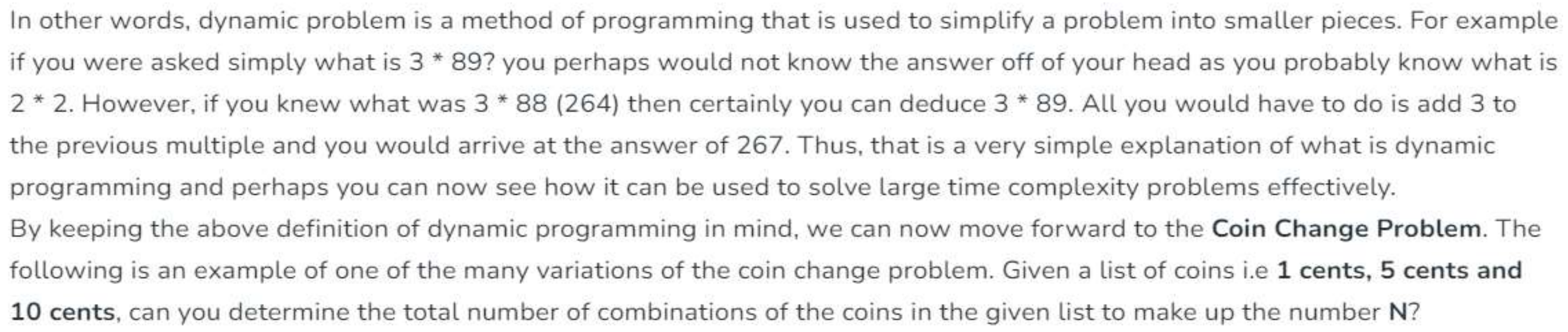
Amypo Technologies Pvt Ltd

## Concept:

- Coin Change Problem
- Longest Increasing Subsequences

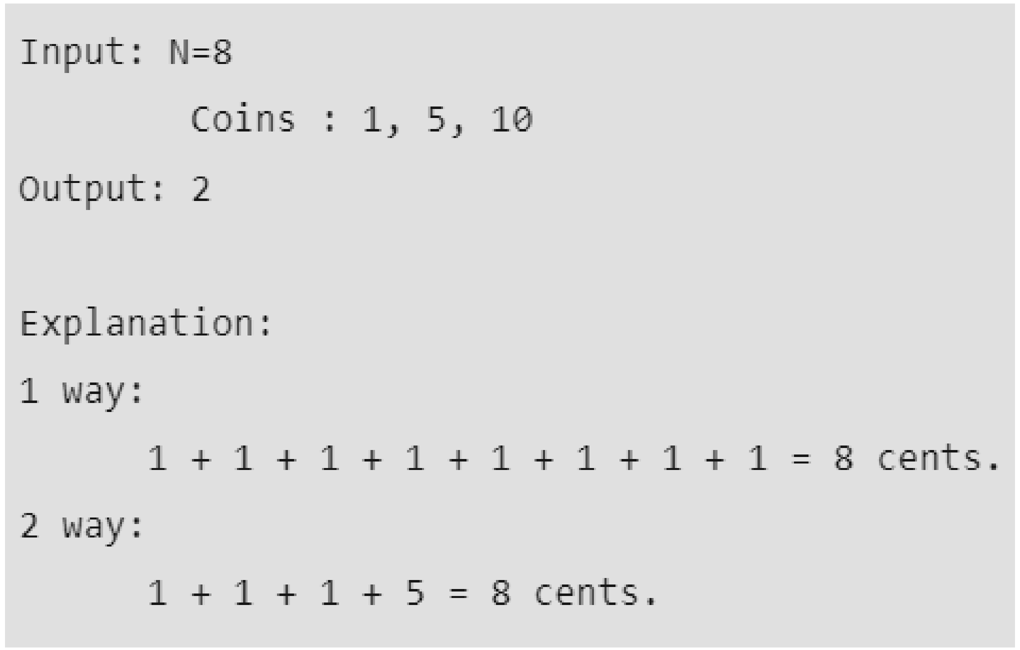
## Coin Change Problem

- The **Coin Change Problem** is considered by many to be essential to understanding the paradigm of programming known as dynamic programming.
- The two often are always paired together because the coin change problem encompass the concepts of dynamic programming. For those who don’t know about dynamic programming



In other words, dynamic problem is a method of programming that is used to simplify a problem into smaller pieces. For example if you were asked simply what is 3 * 89? you perhaps would not know the answer off of your head as you probably know what is 2 * 2. However, if you knew what was 3 * 88 (264) then certainly you can deduce 3 * 89. All you would have to do is add 3 to the previous multiple and you would arrive at the answer of 267. Thus, that is a very simple explanation of what is dynamic programming and perhaps you can now see how it can be used to solve large time complexity problems effectively.
By keeping the above definition of dynamic programming in mind, we can now move forward to the **Coin Change Problem**. The following is an example of one of the many variations of the coin change problem. Given a list of coins i.e **1 cents, 5 cents and 10 cents**, can you determine the total number of combinations of the coins in the given list to make up the number **N**?

**Example 1**: Suppose you are given the coins 1 cent, 5 cents, and 10 cents with N = 8 cents, what are the total number of combinations of the coins you can arrange to obtain 8 cents.



```
Input: N=8
        Coins : 1, 5, 10
Output: 2

Explanation:
1 way:
      1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8 cents.
2 way:
      1 + 1 + 1 + 5 = 8 cents.
```

- All you’re doing is determining all of the ways you can come up with the denomination of 8 cents. Eight 1 cents added together is equal to 8 cents.
- Three 1 cent plus One 5 cents added is 8 cents. So there are a total of 2 ways given the list of coins 1, 5 and 10 to obtain 8 cents

## Longest Increasing Subsequences

- The longest common subsequence (LCS) is defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

- If S1 and S2 are the two given sequences then, Z is the common subsequence of S1 and S2 if Z is a subsequence of both S1 and S2. Furthermore, Z must be a **strictly increasing sequence** of the indices of both S1 and S2.

In a strictly increasing sequence, the indices of the elements chosen from the original sequences must be in ascending order in Z.

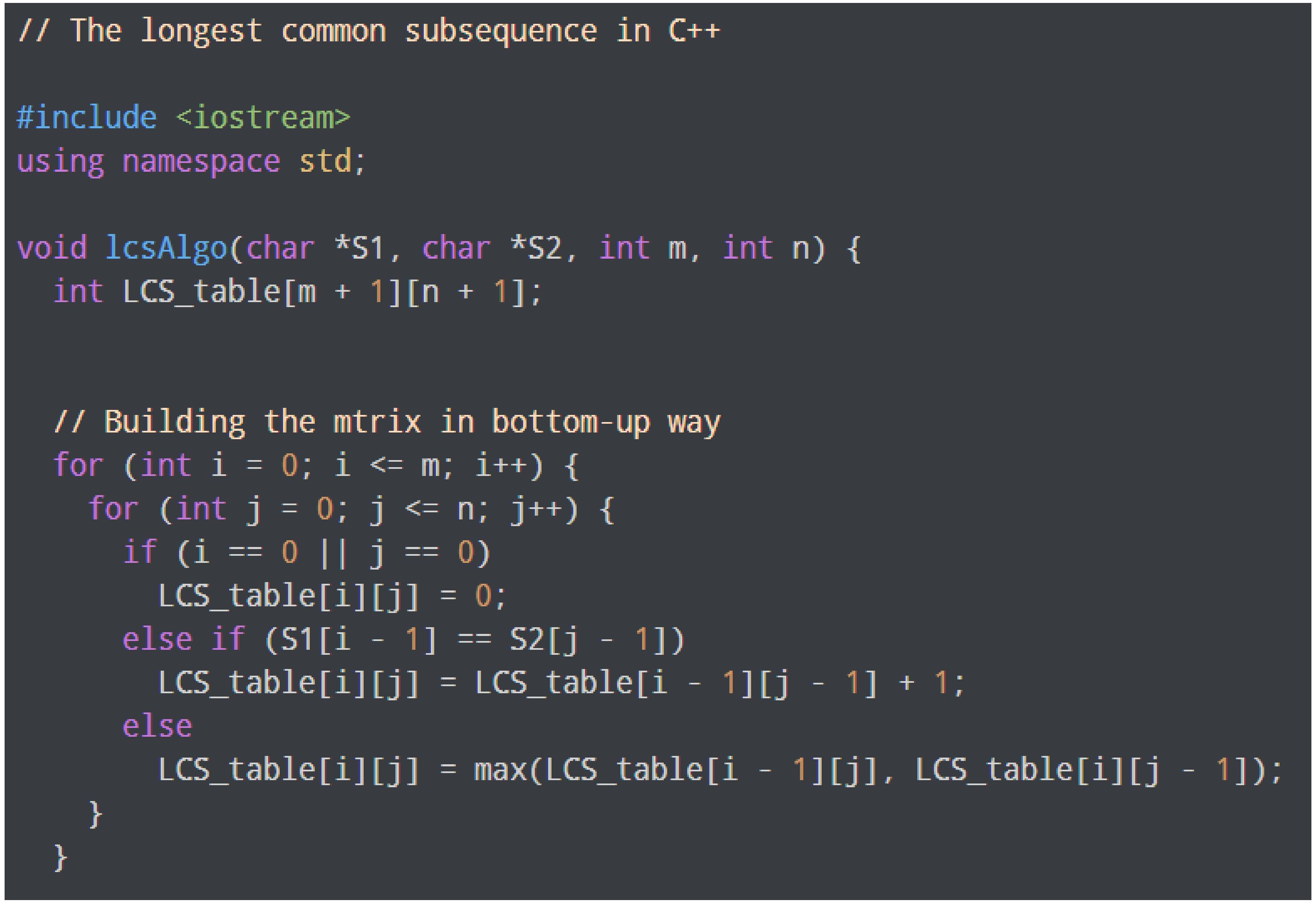
```
S1 = {B, C, D, A, A, C, D}
```

Then, {A, D, B} cannot be a subsequence of S1 as the order of the elements is not the same (ie. not strictly increasing sequence).

```
S1 = {B, C, D, A, A, C, D}
S2 = {A, C, D, B, A, C}
```

- Then, common subsequences are {B, C}, {C, D, A, C}, {D, A, C}, {A, A, C}, {A, C}, {C, D}, ...
- Among these subsequences, {C, D, A, C} is the longest common subsequence. We are going to find this longest common subsequence using dynamic programming.
- Before proceeding further, if you do not already know about dynamic programming, please go through dynamic programming

## Example:



```cpp
// The longest common subsequence in C++

#include <iostream>
using namespace std;

void lcsAlgo(char *S1, char *S2, int m, int n) {
  int LCS_table[m + 1][n + 1];


  // Building the mtrix in bottom-up way
  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      if (i == 0 || j == 0)
        LCS_table[i][j] = 0;
      else if (S1[i - 1] == S2[j - 1])
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      else
        LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
    }
  }
  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';

  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
      lcsAlgo[index - 1] = S1[i - 1];
      i--;
      j--;
      index--;
    }

    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
      j--;
  }

  // Printing the sub sequences
  cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo << "\n";
}
```

```
int main() {
  char S1[] = "ACADB";
  char S2[] = "CBDA";
  int m = strlen(S1);
  int n = strlen(S2);

  lcsAlgo(S1, S2, m, n);
}
```