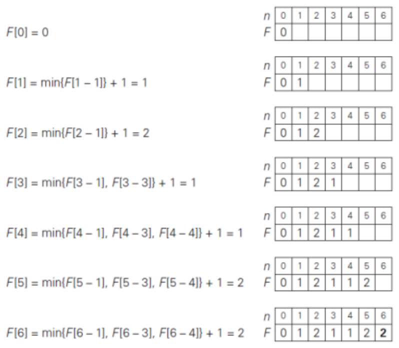
FSM FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ	Mühendislik Fakültesi Bilgisayar Mühendisliği Programı **BLM307E Algorithm Analysis & Design Lab Work**	Adı Soyadı: Notu:

Lab 10: Dynamic Programming

Question 1: Consider the general instance of the following well-known problem. Give change for amount n using the minimum number of coins of denominations d1<d2 < . . .<dm. For the coin denominations used in the United States, as for those used in most if not all other countries, there is a very simple and efficient algorithm discussed in the next chapter. Here, we consider a dynamic programming algorithm for the general case, assuming availability of unlimited quantities of coins for each of the m denominations d1< d2 < . . . < dm where d1 = 1. Implement the java code using the following pseudo-code and the backtrace algorithm that prints the used coin set **(70pts)**.

```
ALGORITHM  ChangeMaking(D[1..m], n)
    //Applies dynamic programming to find the minimum number of coins
    //of denominations $d_1 < d_2 < \cdots < d_m$ where $d_1 = 1$ that add up to a
    //given amount $n$
    //Input: Positive integer $n$ and array $D[1..m]$ of increasing positive
    //        integers indicating the coin denominations where $D[1] = 1$
    //Output: The minimum number of coins that add up to $n$
    $F[0] \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
        $temp \leftarrow \infty$;  $j \leftarrow 1$
        while $j \le m$ and $i \ge D[j]$ do
            $temp \leftarrow \min(F[i - D[j]], temp)$
            $j \leftarrow j + 1$
        $F[i] \leftarrow temp + 1$
    return $F[n]$
```



$F[0] = 0$

n	0	1	2	3	4	5	6
F	0						

$F[1] = \min\{F[1-1]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1					

$F[2] = \min\{F[2-1]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2				

$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	**2**

FIGURE 8.2 Application of Algorithm *MinCoinChange* to amount $n = 6$ and coin denominations 1, 3, and 4.

Expected Output:

```
Denominators = {1, 3, 5}
Amount = 9

Minimum number of coin:3
Coin set:1 3 5
```

```
public static int[] solve(int[] D, int n) {
        int F[] = new int[n+1];
        int temp;
        int j;
        F[0]=0;

        for(int i=1; i<=n; i++){

            temp=Integer.MAX_VALUE;
            j=0;
            while(j<=D.length-1 && i>=D[j]){
                if(F[i-D[j]]<=temp){
                    temp=F[i-D[j]];
                }
                j++;

            }
            F[i]=temp+1;
        }
        return F;

    }
```

	Mühendislik Fakültesi Bilgisayar Mühendisliği Programı **BLM307E Algorithm Analysis & Design Lab Work**	Adı Soyadı: Notu:

```
public static void backtrace(int[] F, int[] D, int amount) {
    for(int i = 0; i <= amount; i++) {
        for(int j = 0; j < F.length; j++) {
            if(F[j] <= i) {
                D[i] = Math.min(D[i], 1 + D[i - F[j]]);
            }
        }

}
```

Q2: What are the space and time efficiency of this algorithm? Use non-recursive analysis to calculate the time efficiency of this algorithm **(30pts)**.

Time efficiency: O(M*sum)
Space a.: O(M*sum)

Q3: A cutting factory purchases large metal poles, cuts them into shorter shafts, and then sells them. Each slice is free of charge. The price is determined by the length of the cut. The manufacturer wishes to optimise its earnings by determining the optimal cuts. Given below are the prices that the manufacturer can produce for each length of the stick. Determine the greatest income, r_n, that can be generated by slicing the metal poles into pieces and selling them using a bottom-up dynamic programming technique **(Optional: 50pts)**.

Length	1	2	3	4	5	6	7
Price	3	5	6	7	8	11	13

r_7 = ?

21

We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut.
Let cutRod(n) be the required (best possible price) value for a rod of length n. cutRod(n) can be written as follows.
cutRod(n) = max(price[i] + cutRod(n-i-1)) for all i in {0, 1 .. n-1}

```
static int cutRod(int price[], int index, int n)
  {
    // base case
    if (index == 0) {
      return n * price[0];
    }
    // At any index we have 2 options either
    // cut the rod of this length or not cut
    // it
    int notCut = cutRod(price, index - 1, n);
    int cut = Integer.MIN_VALUE;
    int rod_length = index + 1;

    if (rod_length <= n)
      cut = price[index]
```

	Mühendislik Fakültesi Bilgisayar Mühendisliği Programı **BLM307E Algorithm Analysis & Design Lab Work**	Adı Soyadı: Notu:

```
              + cutRod(price, index, n - rod_length);

    return Math.max(notCut, cut);
}

/* Driver program to test above functions */
public static void main(String args[])
{
    int arr[] = { 3,5,6,7,8,11,13};
    int size = arr.length;
    System.out.println("Maximum Obtainable Value is "
                    + cutRod(arr, size - 1, size));
}
```