DESIGN AND ANALYSIS OF ALGORITHMS

KARNAM NITHIN,21BAI1091

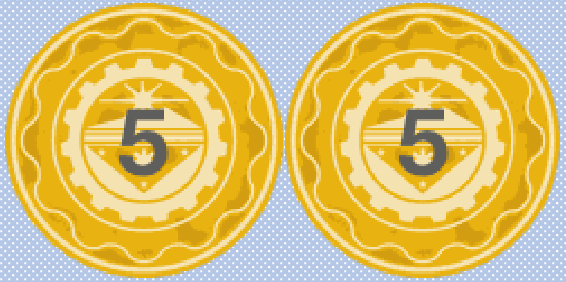
COIN CHANGE ALGORITHM

KARNAM NITHIN,21BAI1091

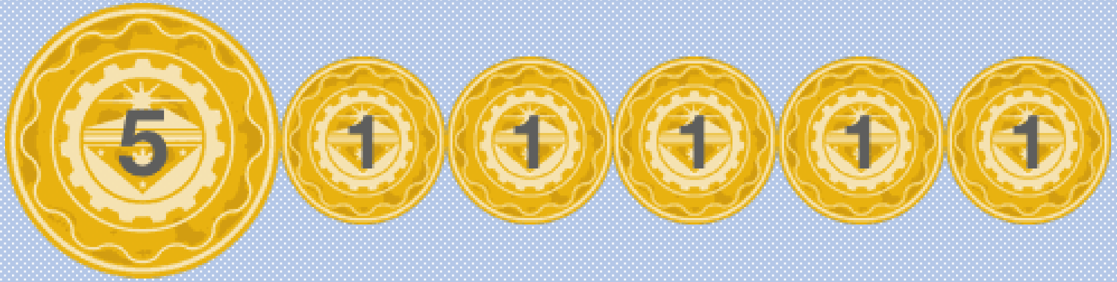
HAVE A GLANCE :

LET US ASSUME WE NEED 10 RS TO BUY A CANDY AND YOU ARE HAVING 12 1 RUPEE COINS,1 10 RUPEE COINS, AND 3 5 RUPEE COINS, FIND WITH WHAT POSSIBLE COMBINATIONS YOU CAN BUY A CANDY?

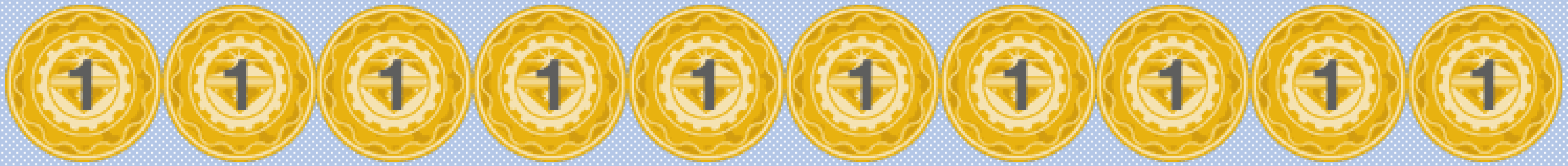
10



10



10



10

KARNAM NITHIN,21BAI1091

DESIGN STRATEGY USED:

DYNAMIC PROGRAMMING(DP)

WHAT IS DP?

A technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as the optimal substructure property. The main use of dynamic programming is to solve optimization problems.

KARNAM NITHIN,21BAI1091

PROBLEM STATEMENT:

?

We are given an array of coins having different denominations and an integer sum representing the total money, you have to return the all possible coins that you will need to make up that sum.

[we have an infinite amount of coins of each type]

LOGIC:

- *Using 2-D vector to store the Overlapping subproblems.*
- *Traversing the whole array to find the solution and storing in the memoization table.*
- *Using the memoization table to find the optimal solution.*

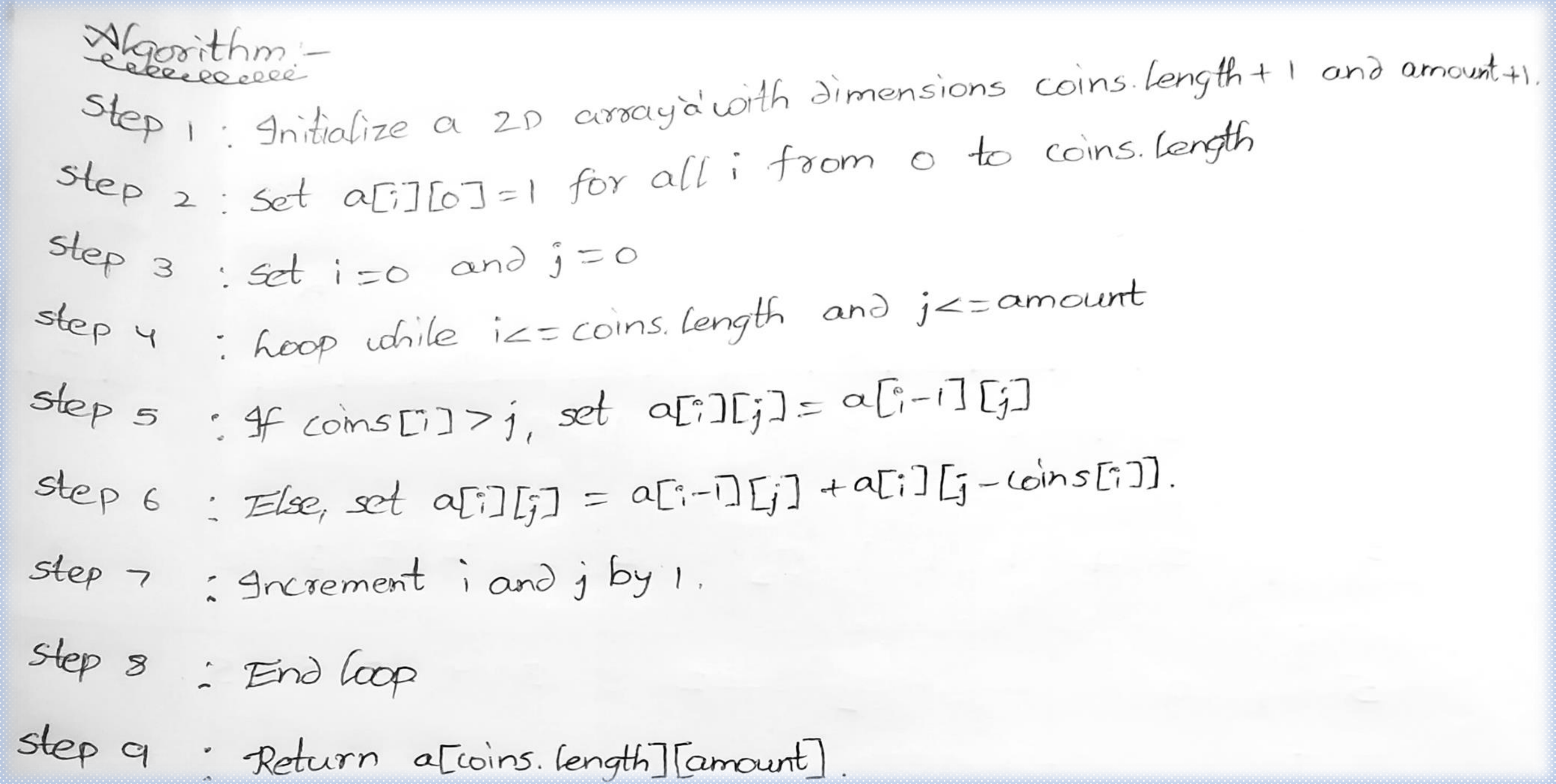
INPUT:

sum = 4,
coins[] = {1,2,3}

OUTPUT:

Output: 4

ALGORITHM:

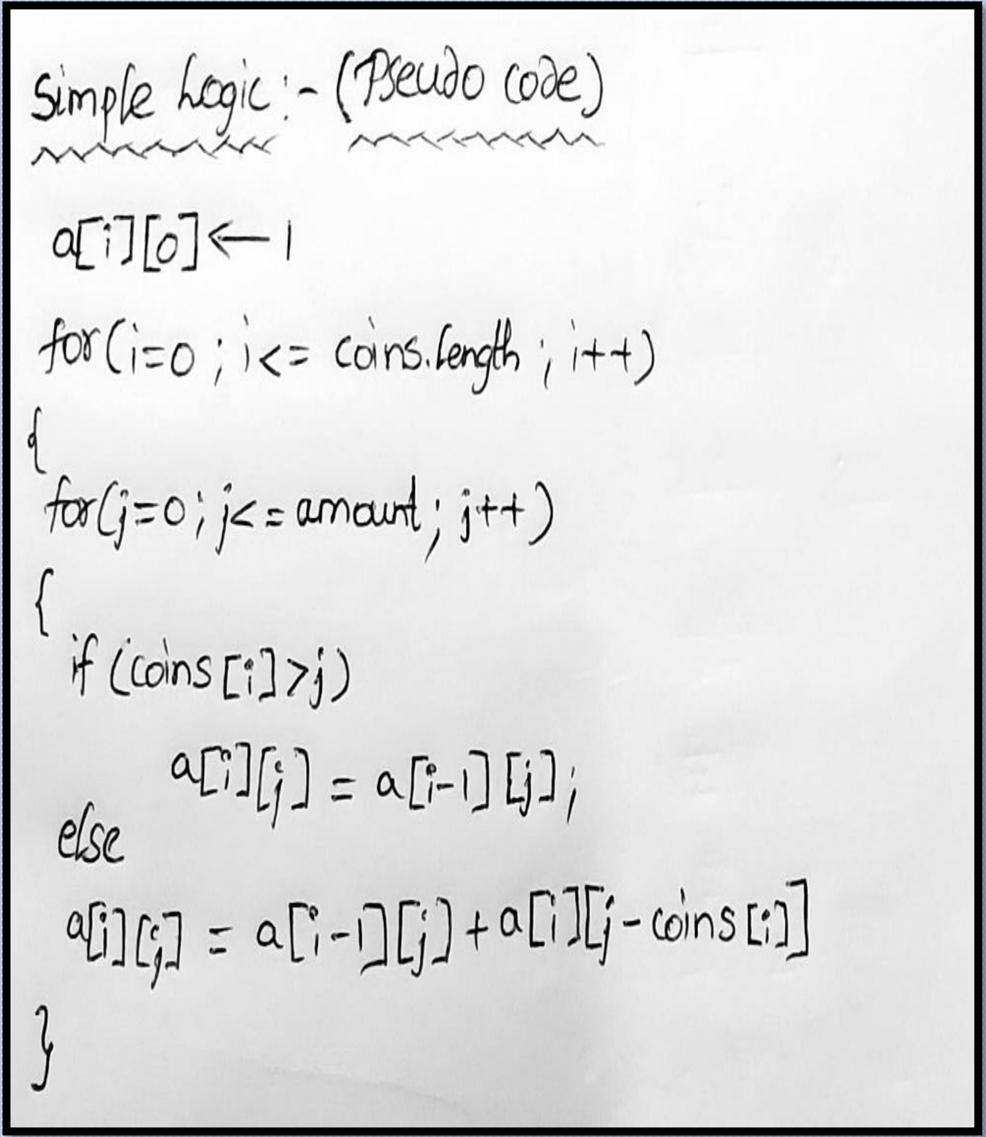


Algorithm :-

```
Step 1 : Initialize a 2D array 'a' with dimensions coins.length + 1 and amount + 1.
Step 2 : Set a[i][0] = 1 for all i from 0 to coins.length
Step 3 : Set i = 0 and j = 0
Step 4 : Loop while i <= coins.length and j <= amount
Step 5 : If coins[i] > j, set a[i][j] = a[i-1][j]
Step 6 : Else, set a[i][j] = a[i-1][j] + a[i][j - coins[i]].
Step 7 : Increment i and j by 1.
Step 8 : End loop
Step 9 : Return a[coins.length][amount].
```

KARNAM NITHIN,21BAI1091

PSEUDOCODE:



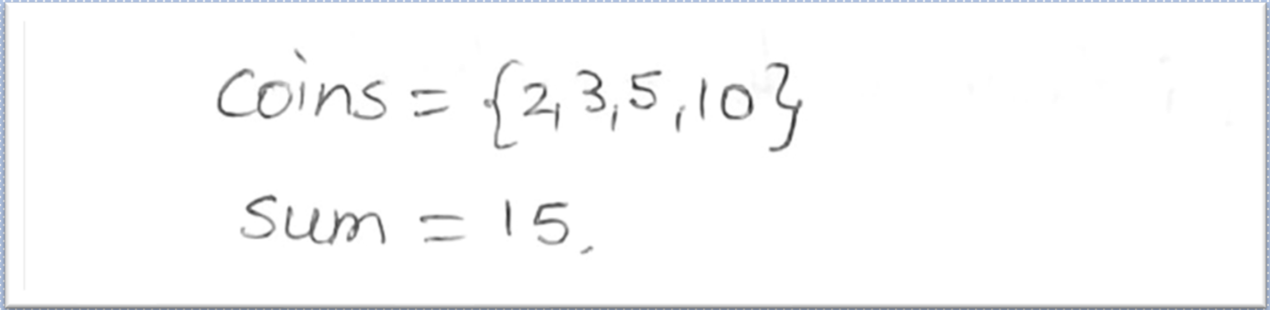
Simple Logic :- (Pseudo code)

```
a[i][0] ← 1
for(i=0 ; i<= coins.length ; i++)
{
for(j=0 ; j<= amount ; j++)
{
  if (coins[i]>j)
          a[i][j] = a[i-1][j];
  else
  a[i][j] = a[i-1][j] + a[i][j-coins[i]]
}
```

KARNAM NITHIN,21BAI1091

ILLUSTRATION:

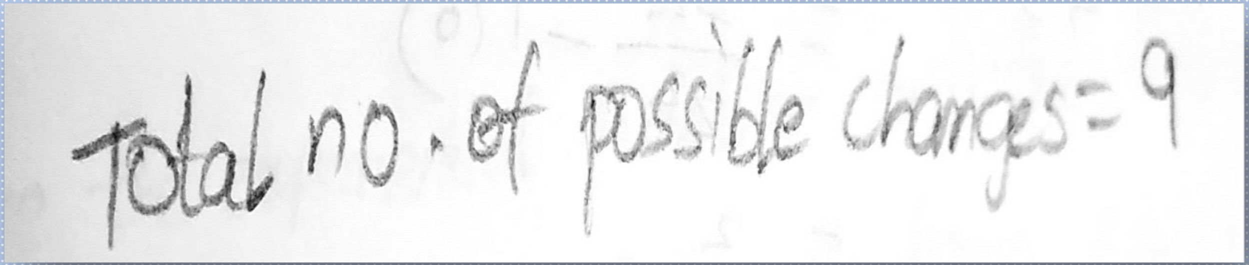
INPUT:



coins = {2,3,5,10}

sum = 15.

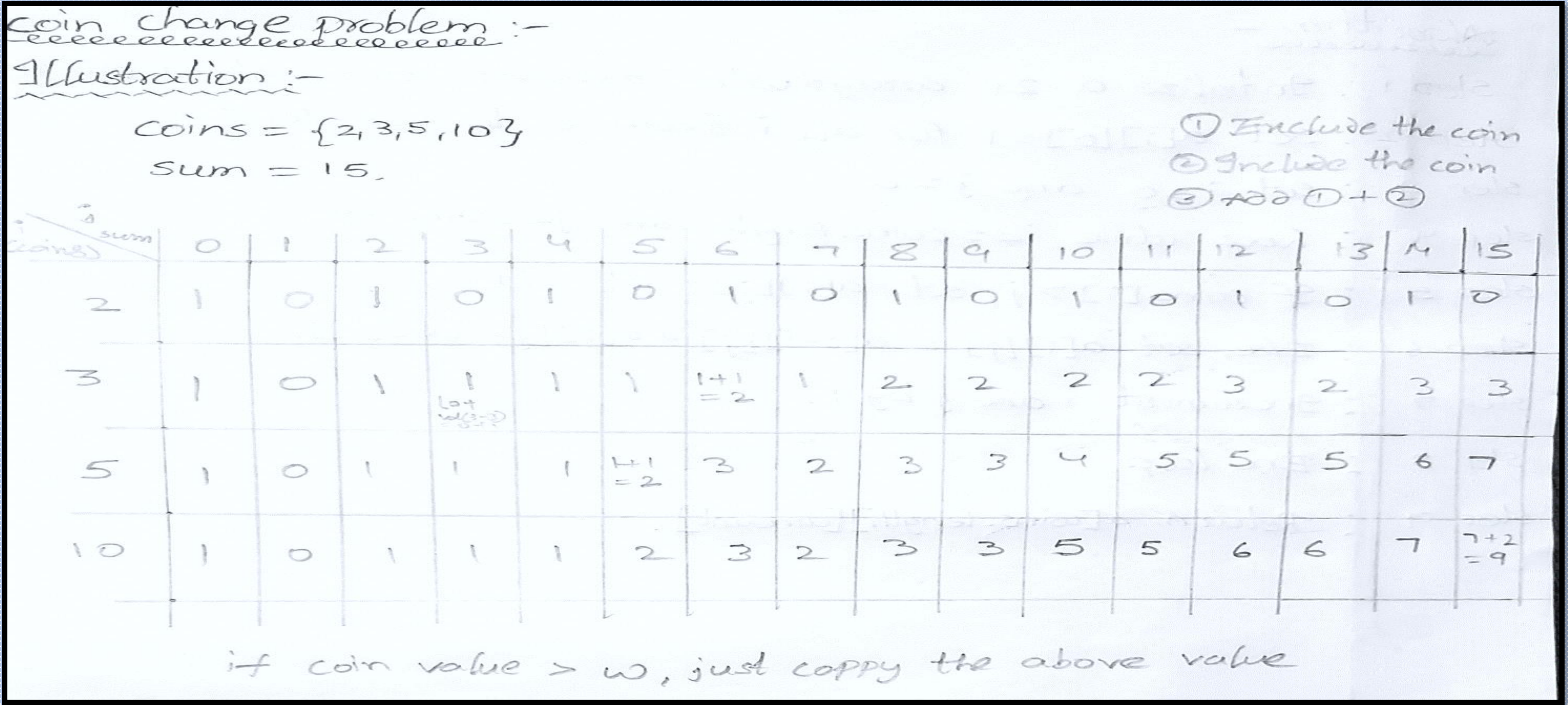
OUTPUT:



Total no. of possible changes = 9

KARNAM NITHIN,21BAI1091

SIMULATION:



Coin change problem :-

Illustration :-

coins = {2, 3, 5, 10}
sum = 15.

① Exclude the coin
② Include the coin
③ Add ① + ②

i (coins) \ j sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
3	1	0	1	1 0+ val(3-3) =0+1	1	1	1+1 =2	1	2	2	2	2	3	2	3	3
5	1	0	1	1	1	1+1 =2	3	2	3	3	4	5	5	5	6	7
10	1	0	1	1	1	2	3	2	3	3	5	5	6	6	7	7+2 =9

if coin value > w, just coppy the above value

PROOF OF CORRECTNESS :

The proof of correctness for the above pseudo code is as follows:

Let A(i,j) be the number of ways to make change for j amount of money using coins[0] to coins[i].We can prove that A(i,j) is correct by induction on i.

Base Case:

When i=0, A(0,j) is the number of ways to make change for j amount of money using only coins[0].We know that if coins[0] > j, then A(0,j) = 0.If coins[0] <= j, then A(0,j) = 1, since the only way to make change for j is to use the coin coins[0] j times.

KARNAM NITHIN,21BAI1091

Inductive Step:

Assume that A(i-1,j) is correct.

We need to prove that A(i,j) is also correct.

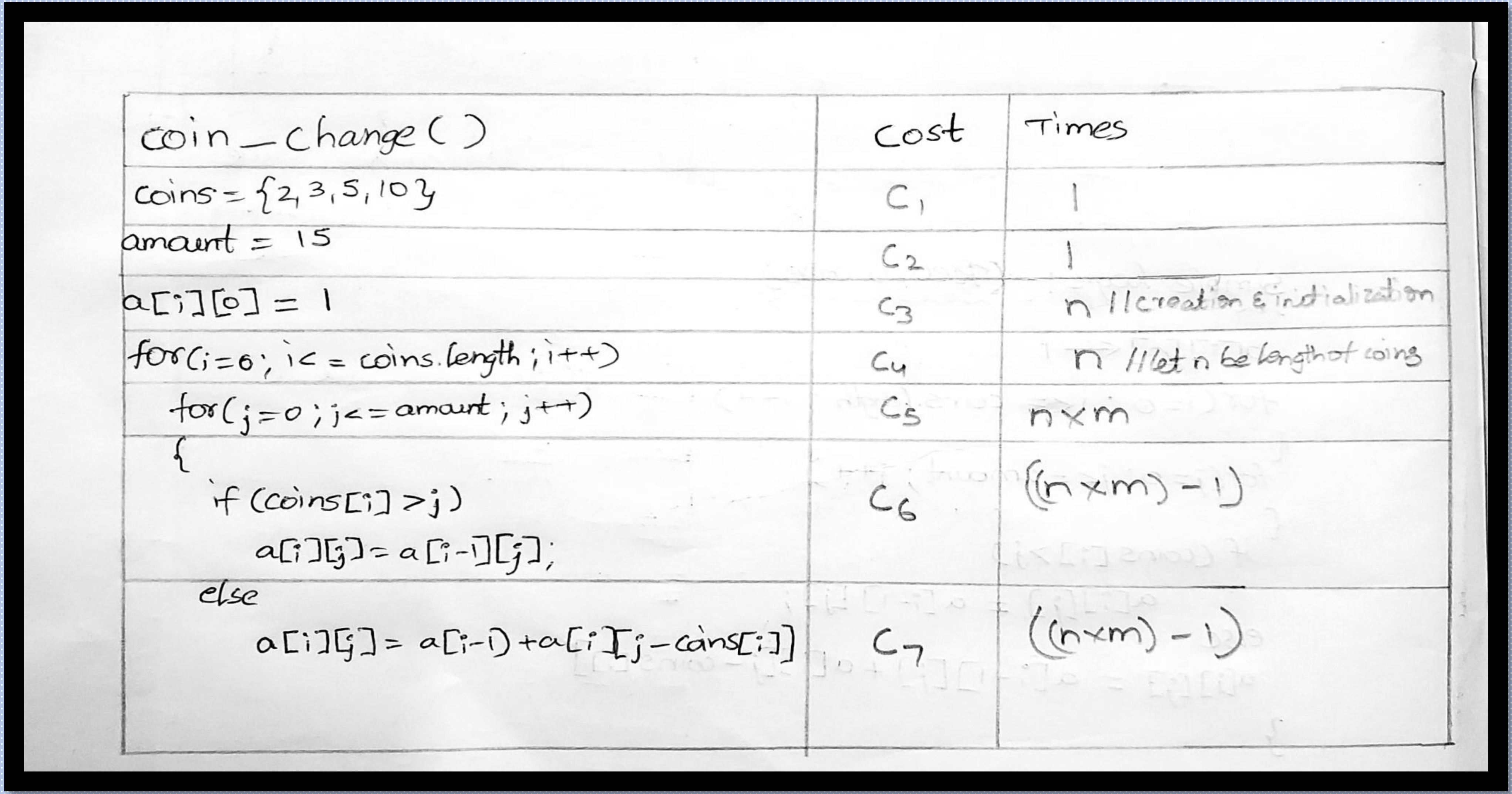
We know that A(i,j) is the number of ways to make change for j amount of money using coins[0] to coins[i]. If coins[i] > j, then the only way to make a change is to use coins[0] to coins[i-1], which is A(i-1,j).

Therefore, A(i,j) = A(i-1,j).If coins[i] <= j, then we can make change either by using coins[0] to coins[i-1] (A(i-1,j)) or by using coins[i] in addition to coins[0] to coins[i-1] (A(i,j-coins[i])).

Therefore, A(i,j) = A(i-1,j) + A(i,j-coins[i]).

This completes the proof that A(i,j) is correct.

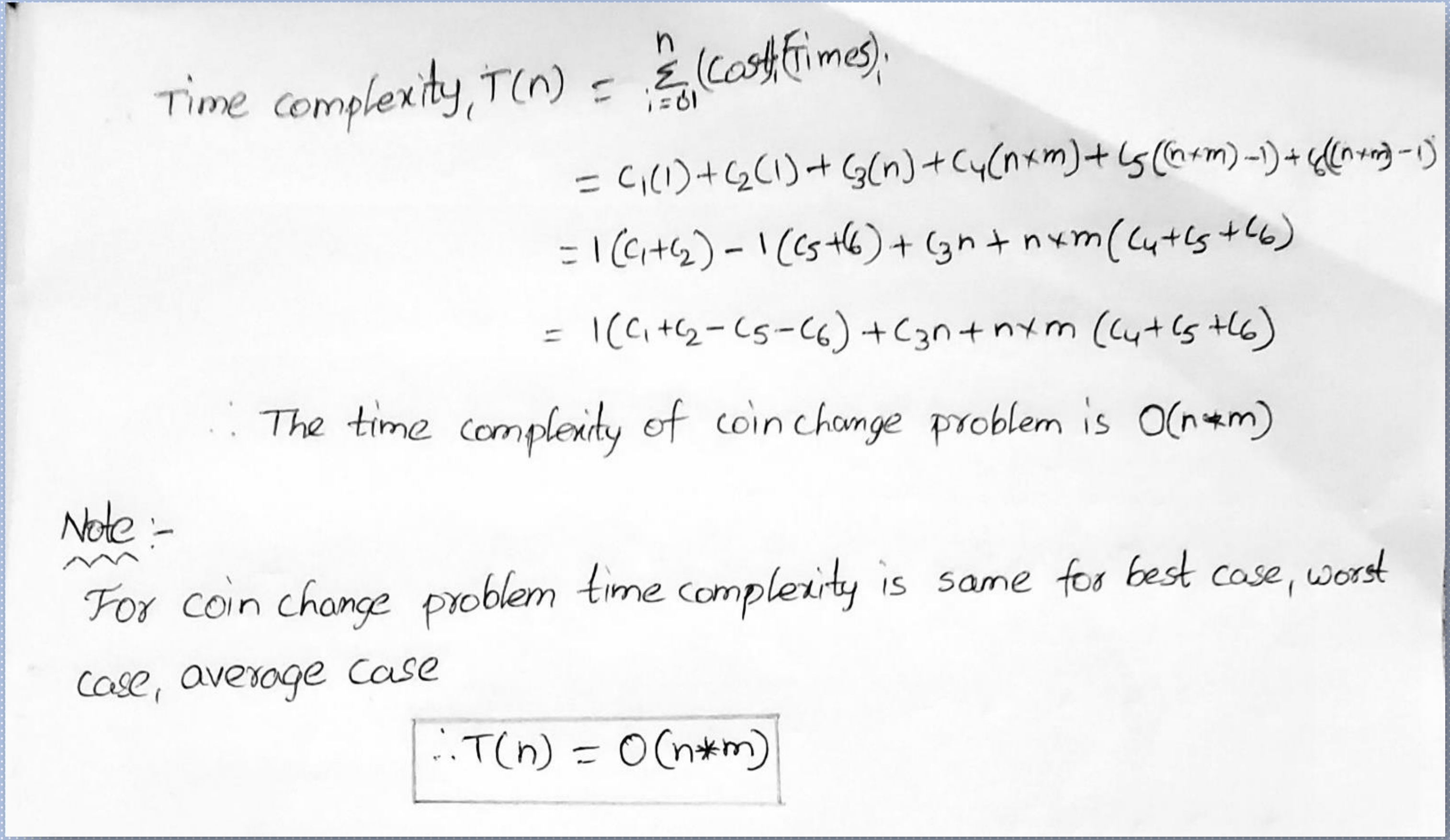
COIN CHANGE ANALYSIS:



coin_change( )	Cost	Times
coins = {2,3,5,10}	C_1	1
amount = 15	C_2	1
a[i][0] = 1	C_3	n //creation & initialization
for(i=0; i<= coins.length; i++)	C_4	n //let n be length of coins
for(j=0; j<=amount; j++)	C_5	n×m
{ if (coins[i] >j) a[i][j] = a[i-1][j];	C_6	((n×m) - 1)
else a[i][j] = a[i-1] + a[i][j-coins[i]]	C_7	((n×m) - 1)

KARNAM NITHIN,21BAI1091

TIME COMPLEXITY CALCULATION:



Time complexity, $T(n) = \sum_{i=01}^{n} (\text{cost})_i (\text{Times})_i$

$$= C_1(1) + C_2(1) + C_3(n) + C_4(n \times m) + C_5((n \times m) - 1) + C_6((n \times m) - 1)$$

$$= 1(C_1 + C_2) - 1(C_5 + C_6) + C_3 n + n \times m (C_4 + C_5 + C_6)$$

$$= 1(C_1 + C_2 - C_5 - C_6) + C_3 n + n \times m (C_4 + C_5 + C_6)$$

$\therefore$ The time complexity of coin change problem is $O(n*m)$

Note :-

For coin change problem time complexity is same for best case, worst case, average case

$$\boxed{\therefore T(n) = O(n*m)}$$

KARNAM NITHIN,21BAI1091

WHERE THIS ALGORITHM IS USED?

1. Currency Exchange: Using the coin change problem, it is possible to create an algorithm to determine the optimal combination of coins to use when converting between two currencies.
2. Automated Vending Machines: Vending machines use the coin change problem to determine the optimal combination of coins to return when a customer purchases an item.
3. Coin Counting Machines: Coin counting machines use the coin change problem to determine the optimal combination of coins to return when a customer pays for their purchase.
4. Shopping Carts: Shopping carts can use the coin change problem to determine the optimal combination of coins to return when a customer pays for their purchase.

KARNAM NITHIN,21BAI1091

REFERENCES:

Algorithms in a nutshell

- George T. Heineman

Algorithms Unlocked

- Thomas H. Cormen

https://www.geeksforgeeks.org/coin-change-dp-7/

https://www.simplilearn.com/tutorials/data-structure-tutorial/coin-change-problem-with-dynamic-programming

☺ THANK YOU ☺

KARNAM
NITHIN
21BAI1091