

## TASK 1 :- / (Home Task 1)

Time Complexity

## → Iterative

The total number of iteration is  $n-1$ , leading to  $O(n)$  iterations. Each iteration

perform  $O(1)$  so,  $O(n) \times O(1) \Rightarrow \boxed{O(n)}$

## → Recursive

Formula:  $T(n) = T(n-1) + T(n-2) + O(1)$

$$T(0) = O(1), T(1) = O(1)$$

so,  $\Rightarrow \boxed{O(2^n)}$

## → DP (Memoization)

Formula:  $T(n) = T(n-1) + T(n-2)$

$$F(0) = O(1), F(1) = O(1)$$

Unique Subproblem are  $n$  from  $F(0)$  to  $F(n)$

so,  $\Rightarrow \boxed{O(n)}$

## → DP (Tabulation)

array of size  $n+1$ :  $O(n)$

loop runs from 2 to  $n$ :  $n-1$  iterations

work per iteration:  $O(1)$

so,  $\Rightarrow \boxed{O(n)}$

### Task 3:

#### Time Complexity

##### → Iterative

It uses two loops, one for length and one for cutting the rod at each possible point SO,  $\Rightarrow \boxed{O(n^2)}$

##### → Recursive

For each length  $n$ , we try every possible cut, which generates two subproblems making the time complexity ~~exponential~~ exponential SO,  $\Rightarrow \boxed{O(2^n)}$

##### → Memoization

Since each subproblem is solved once and stored in the memo array and for each subproblem, we run a loop of size  $n$ , the time complexity becomes quadratic SO,  $\Rightarrow \boxed{O(n^2)}$

##### → Tabulation

For each length  $i$  from 1 to  $n$ , we run a nested loop of size  $i$ ,

SO,  $\Rightarrow \boxed{O(n^2)}$



## Task 2: / (Home Task 3)

### Time Complexity:

#### → Iterative

• Iterate each character in  $S1$ :  $O(m)$   
For each character in  $S1$ , it iterates through each character in  $S2$  which is  $O(n)$ .

• The table dp of size  $m \times n$  is filled which involves as comparison of characters and updating table.

So,  $\boxed{O(m \times n)}$

#### → Recursive

This approach explores every possible substring and for each comparison it generates three recursive calls. The recursive depth can go up to  $O(m+n)$ .

• The number of recursive calls grows exponentially, making the total time complexity very inefficient.

So,  $\boxed{O(3^{\min(m,n)})}$

## → Memoization

The memoized solution ensures that each subproblem is solved only once. The number of unique subproblems is bounded by  $O(m \times n)$ , where  $m$  is length of  $s1$  and  $n$  is the length of  $s2$ .

So,  $\boxed{O(m \times n)}$

## → Tabulation

Similar to the iterative approach, we fill a table of size  $m \times n$ . The outer loop runs  $O(m)$  times, and for each iteration of the outer loop, the inner loop runs  $O(n)$  times.

So,  $\boxed{O(m \times n)}$

---



## Home Task 2:

### Time Complexity

#### → Iterative

This case contain loops one is Outer loop which run  $\text{len}(\text{coins})$  times and inner loop that run approx. target times for each coin. So,

$$O(\text{len}(\text{coin}) \times \text{Target}) = \boxed{O(m \times n)}$$

#### → Recursive

In Recursive technique, we make a recursive call for each coin for the remaining amount ( $\text{target} - \text{coin}$ ). At each level of recursion, the problem size decreases by one of the coin denominations, leading to an exponential

number of calls So,  $\boxed{O(m^n)}$

$m \Rightarrow$  number of coin

$n \Rightarrow$  target amount

#### → Memoization

In this approach we store the result of each target in a memoization dictionary,

ensuring that each subproblem is solved only once. For each subproblem we loop

over the coins ( $m$  coins) and make recursive calls. Since we only solve each subproblem once and store it in the memo, the number of unique subproblems is  $n$  (equal to target value). So,  $\boxed{O(m \times n)}$   $m \Rightarrow$  number of coins  
 $n \Rightarrow$  Target amount

### → Tabulation

In this approach, we build the solution from the ground up using a table  $[dp[]]$ . For each amount  $i$  from 1 to target we loop through all  $m$  coins. The total number of iterations is proportional to  $m * n$  (i.e. for  $i$  we check  $m$  coins) So,  $\boxed{O(m \times n)}$

## Home Task 4:

### Time Complexity

#### → Iterative

In this approach, there is only one loop that runs  $n$  times and each iteration contains constant time.

So,  $\boxed{O(n)}$



### → Recursive

This approach has an exponential time complexity due to repeated calculations of the same subproblems. The time complexity is  $O(2^n)$

### → Memoization

Each subproblem is computed only once and stored in a dictionary.

So,  $O(n)$  there are  $n$  subproblems.

### → Tabulation

The number of operations is proportional to  $n$ , as each subproblem is solved in a loop, thus time complexity is  $O(n)$

## Home Task 5:

### Time Complexity

#### → Iterative

In the iterative approach the time

Complexity is  $[O(n \times w)]$  because we iterate all over items and all capacities.

### → Recursive

The time complexity in recursive technique is exponential  $[O(2^n)]$  because for each item, we have two choices (include or exclude).

### → Memoization

The time complexity in memoization is  $[O(n \times w)]$  because we solve each subproblem only once.

### → Tabulation

The time complexity is  $[O(n \times w)]$  where  $n$  is the number of items and  $w$  is the knapsack capacity.

---



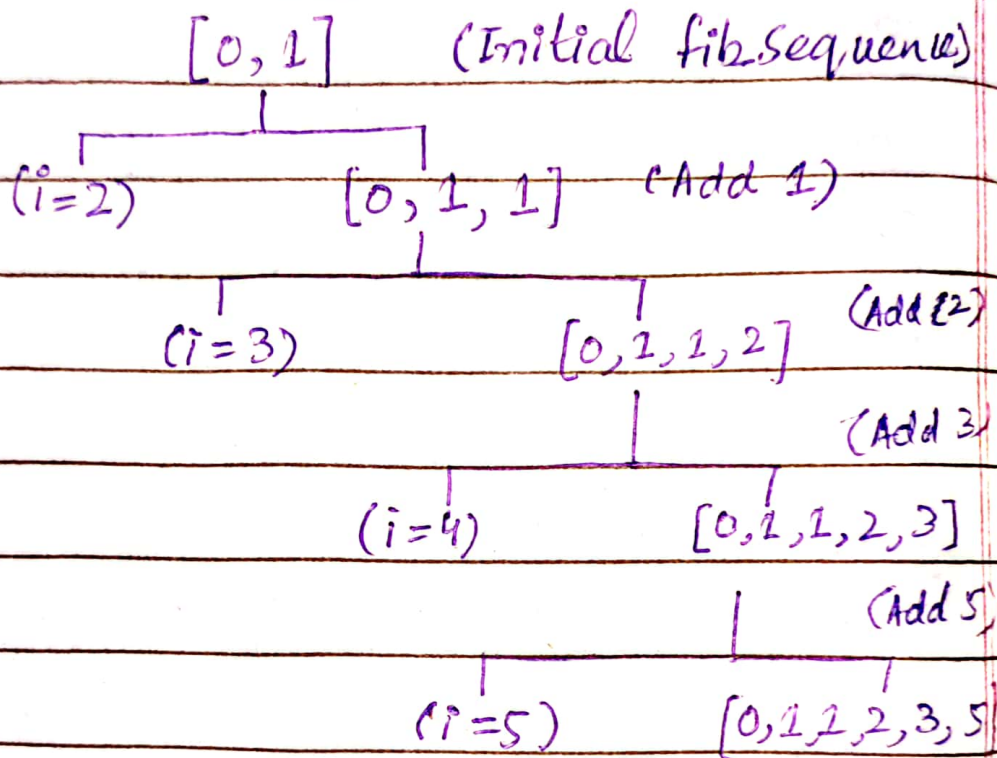
# Dry Run

DATE: \_\_\_/\_\_\_/20\_\_

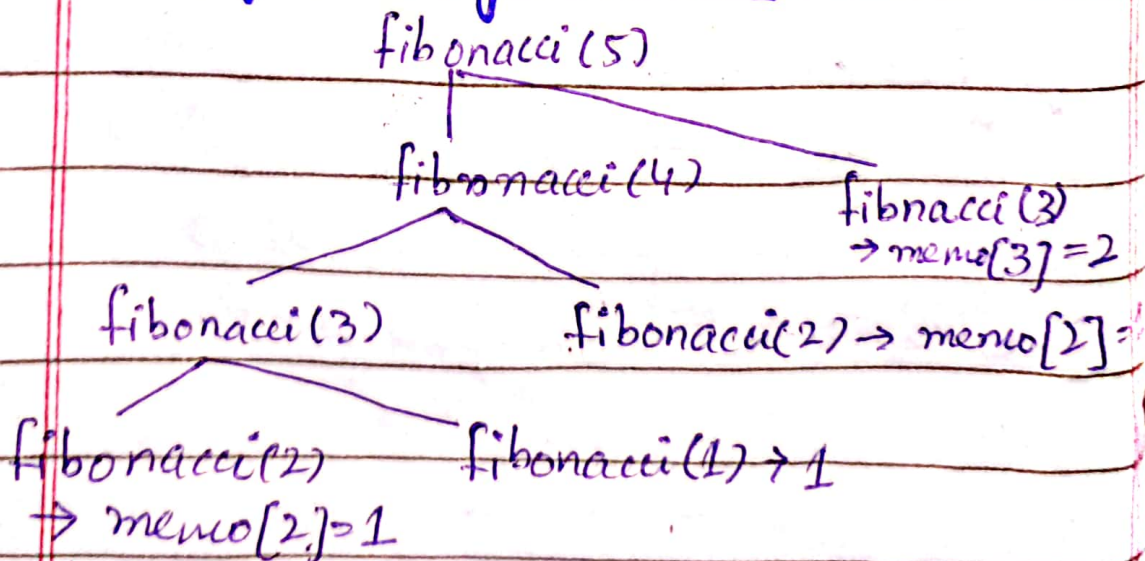
W T W T F S

## Home Task 1:

### Iterative

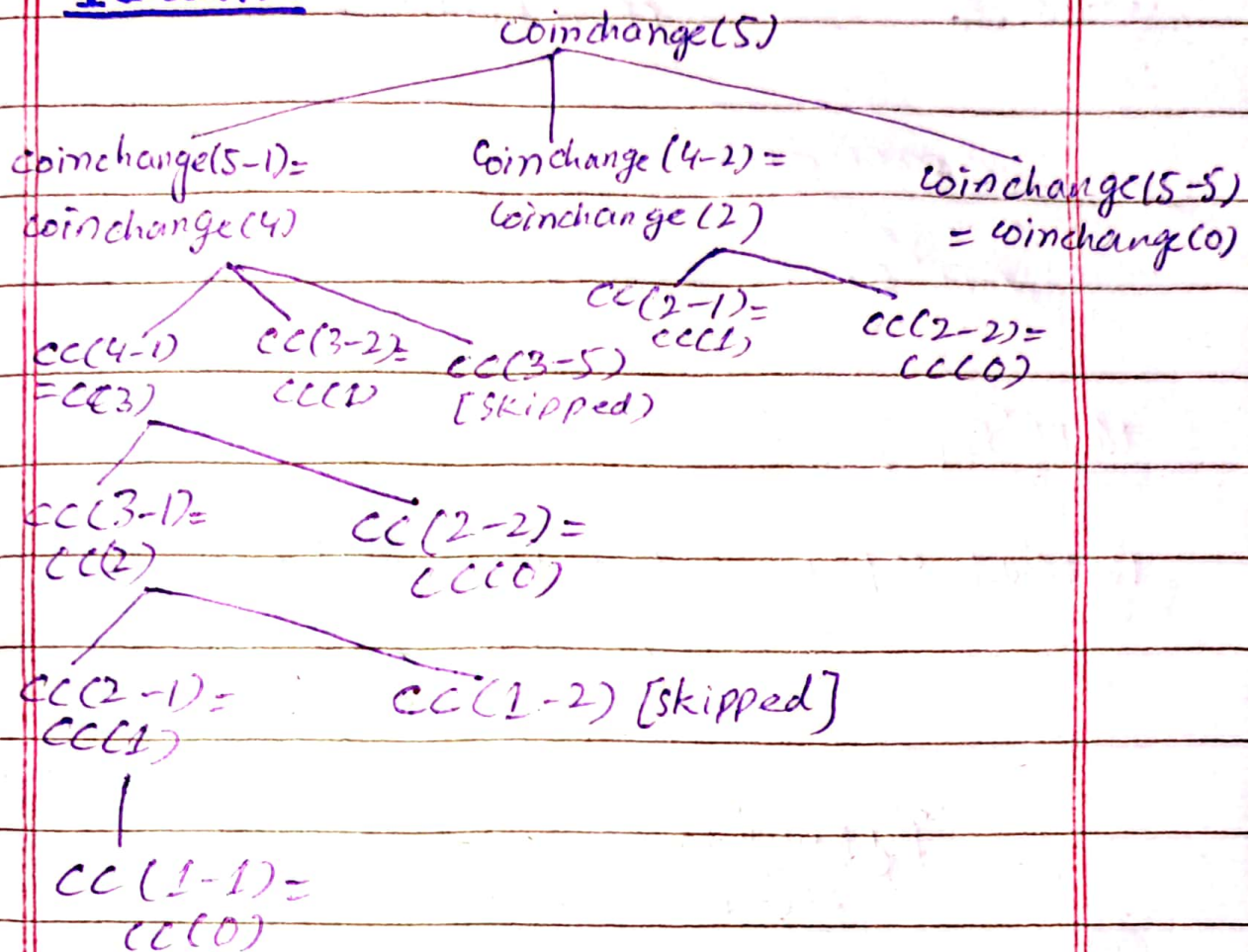
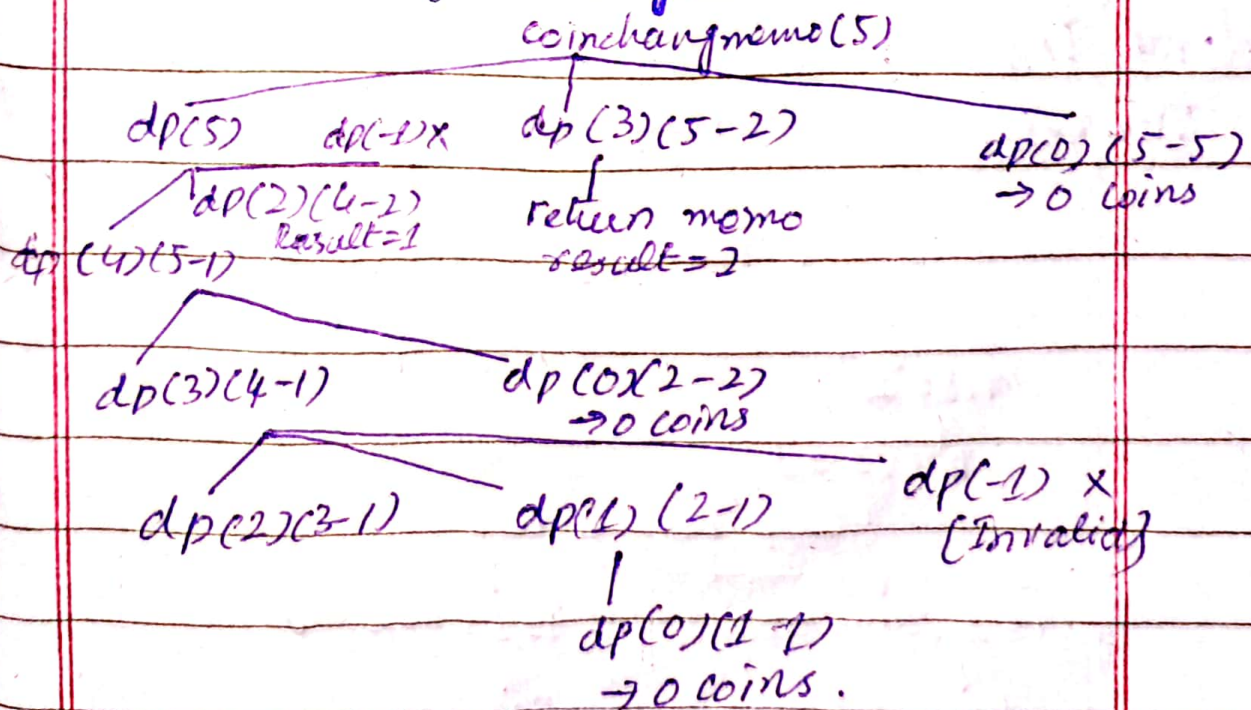


## Dynamic Programming (Memo)

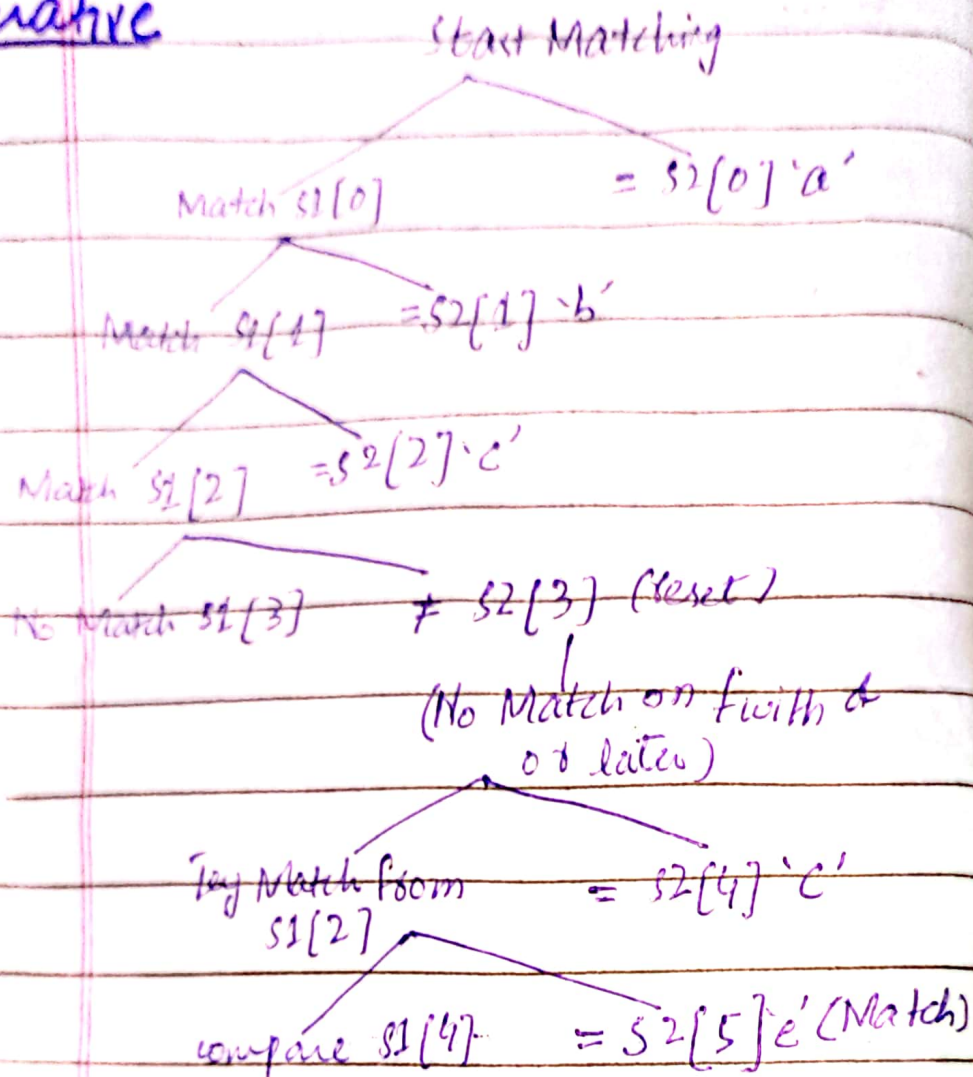
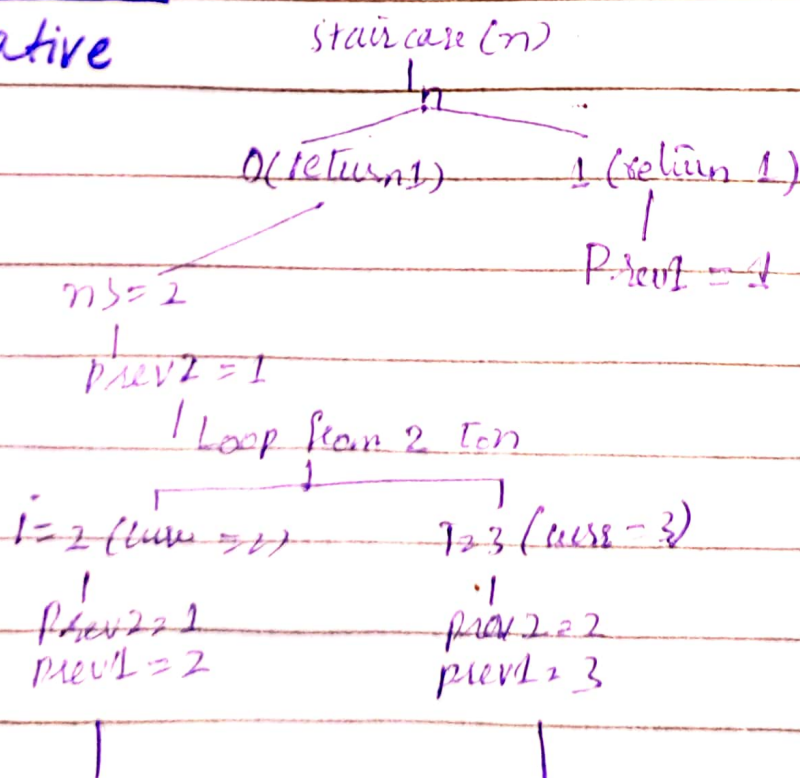


Home Task 2:

[CC = Coinchange]

IterativeDynamic Programming (Memo)



Home Task 3:IterativeHome Task 4:Iterative

DATE: \_\_\_/\_\_\_/20\_\_

M T W T F S

$i=4$  (curr = 5)

prev2 = 2  
prev1 = 3

$i=5$  (curr = 8)

prev2 = 3  
prev1 = 5

Return prev1

## Home Task 5:

### Iterative

knap sack (Values, weight, w)

$n=3$

$dp = [0] * (w+1)$

Loop i from 0 to  $n-1$

$i=0$

$i=1$

$i=2$

update dp[1]

Update dp[2]

return dp[w]