Dynamic Programming

* general, powerful algo design technique. (DP)

① DP is just a "careful brute force"

② DP = subproblems + "reuse"

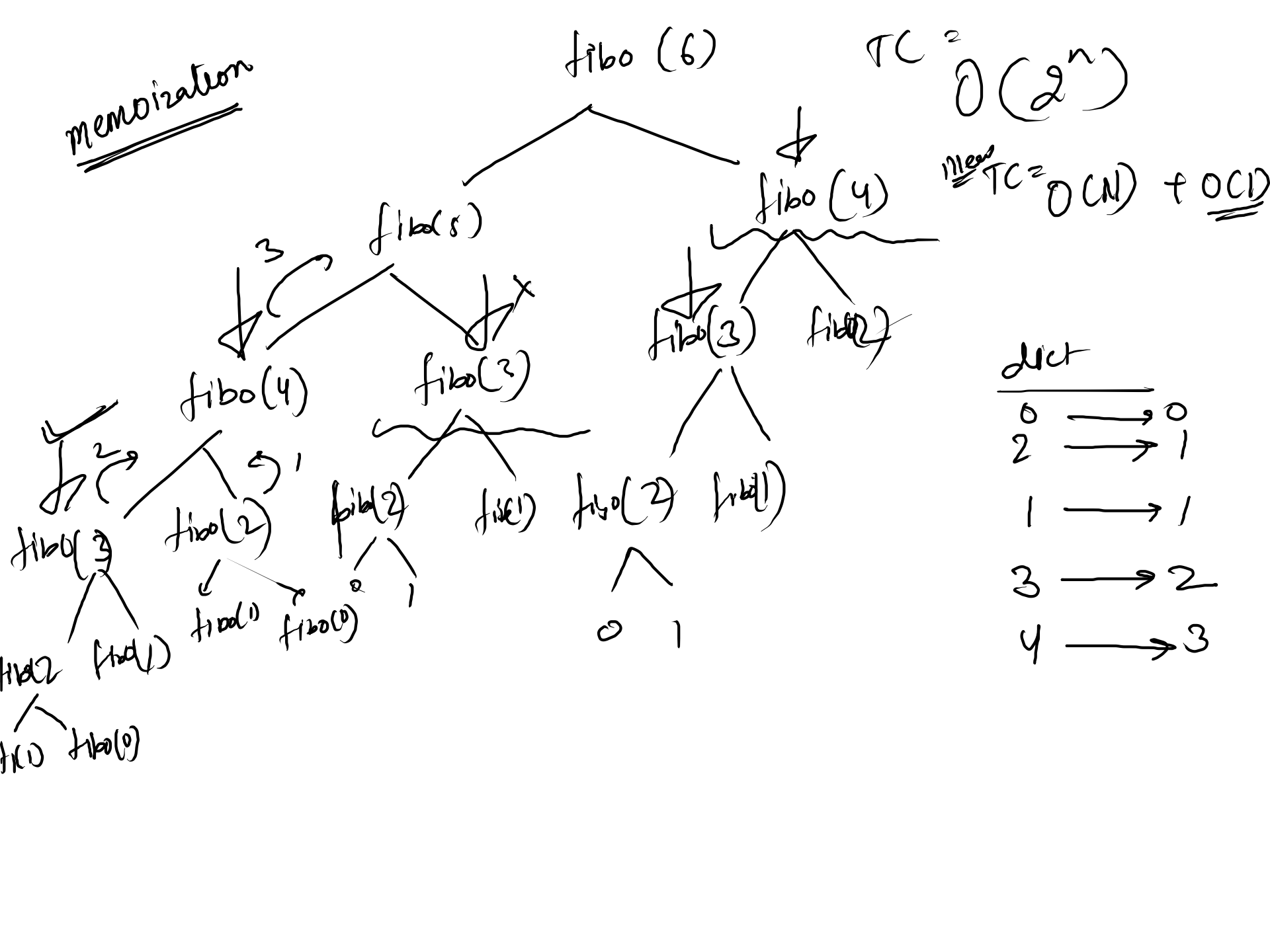
* Those who cannot remember the past are condemned to repeat it.

fibonacci Series

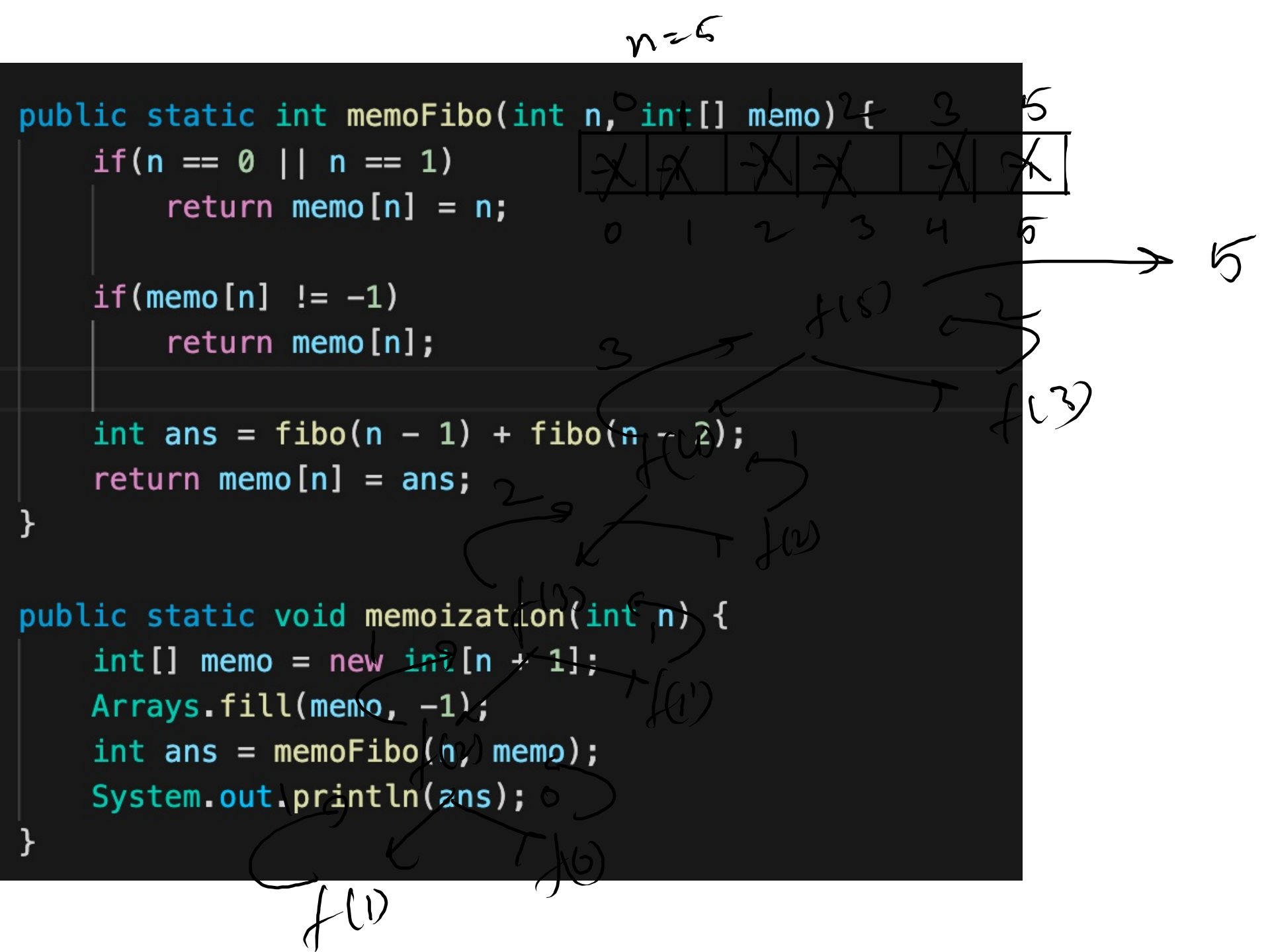
$$\begin{cases} F_1 = 0 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

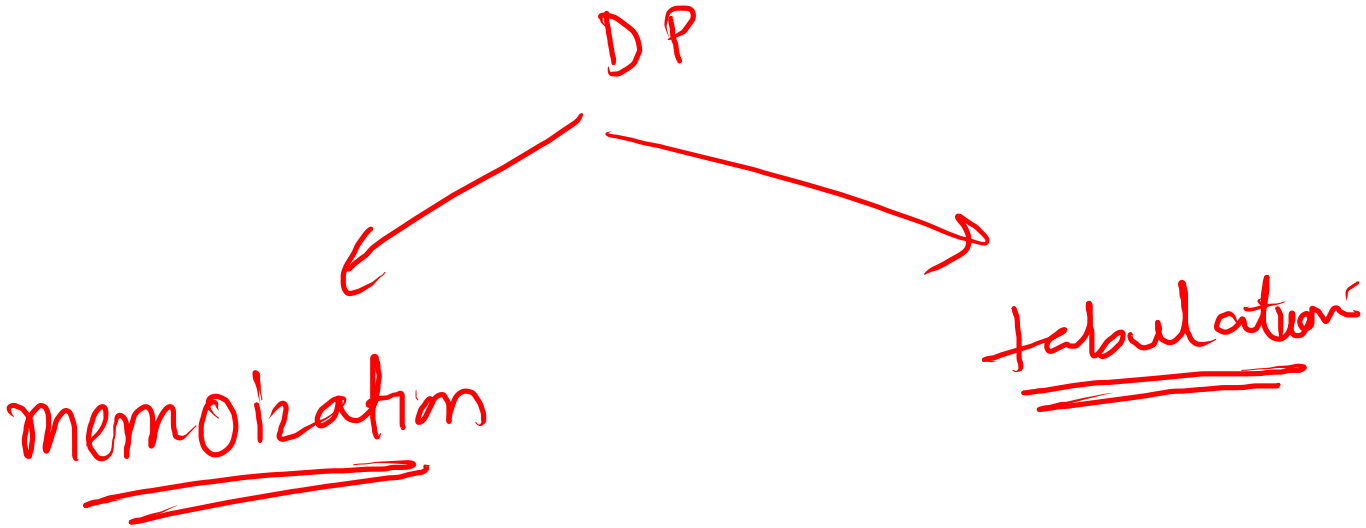
0, 1, 1, 2, 3, 5, 8, 13, - - -

nth fibo


memoization
fibo (6)
TC = O(2^n)
fibo(5)
fibo (4)
Mean TC = O(N) + O(1)
fibo(4)
fibo(3)
fibo(3)
fibo(2)
fibo(2)
fibo(2)
fibo(1)
fibo(2)
fibo(1)
fibo(1)
fibo(0)
dict
0 → 0
2 → 1
1 → 1
3 → 2
4 → 3

```java
public static int memoFibo(int n, int[] memo) {
    if(n == 0 || n == 1)
        return memo[n] = n;

    if(memo[n] != -1)
        return memo[n];

    int ans = fibo(n - 1) + fibo(n - 2);
    return memo[n] = ans;
}

public static void memoization(int n) {
    int[] memo = new int[n + 1];
    Arrays.fill(memo, -1);
    int ans = memoFibo(n, memo);
    System.out.println(ans);
}
```




DP
memoization
tabulation

```java
public static void tabulation(int n) {
    int[] dp = new int[n + 1];
    for (int i = 0; i <= n; i++) {
        if (i == 0 || i == 1) {
            dp[i] = i;
            continue;
        }

        int ans = dp[i - 1] + dp[i - 2];
        dp[i] = ans;
    }
    System.out.println(dp[n]);
}
```

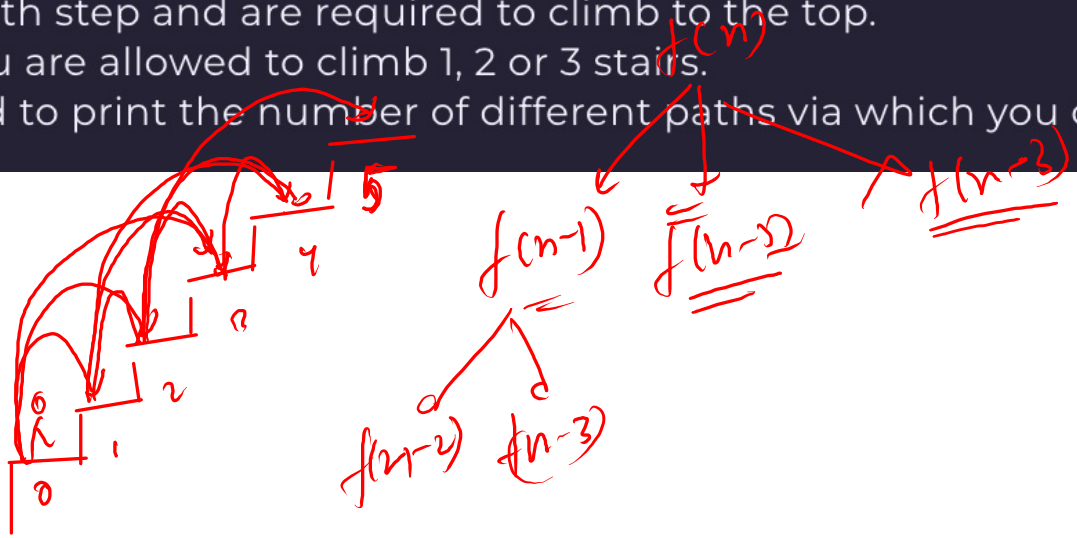
0	1	2	3	4	5	6	7	8
0	1	1	2	3	5	8	13	21

dp

TC = O(N)

SC = O(N)

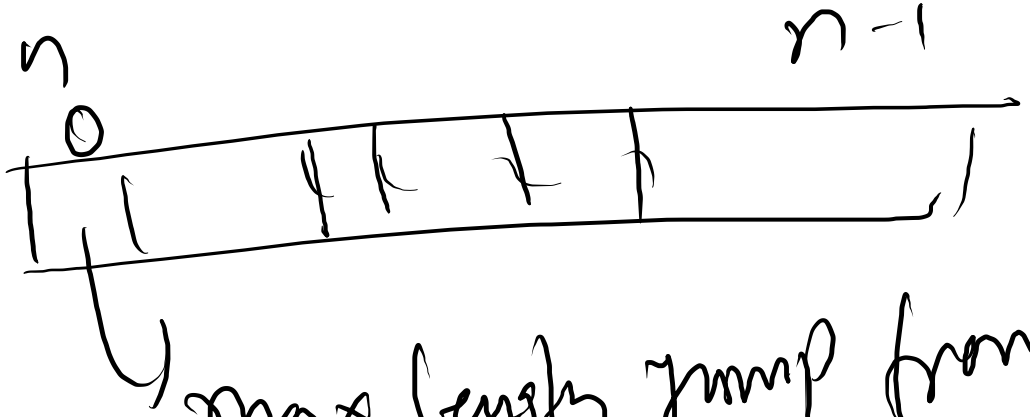
1. You are given a number n, representing the number of stairs in a staircase.
2. You are on the 0th step and are required to climb to the top.
3. In one move, you are allowed to climb 1, 2 or 3 stairs.
4. You are required to print the number of different paths via which you can climb to the top.



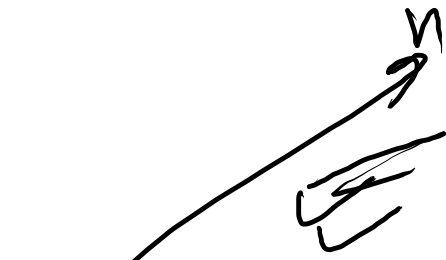
dp

1. Recursive, recurrence relation
2. Memoization
   - 2.1 Store in a dict
   - 2.2 use value already computed
3. Tabulation
   - 3.1 return → continue
   - 3.2 function name → dp array
4. look if space can be optimized.

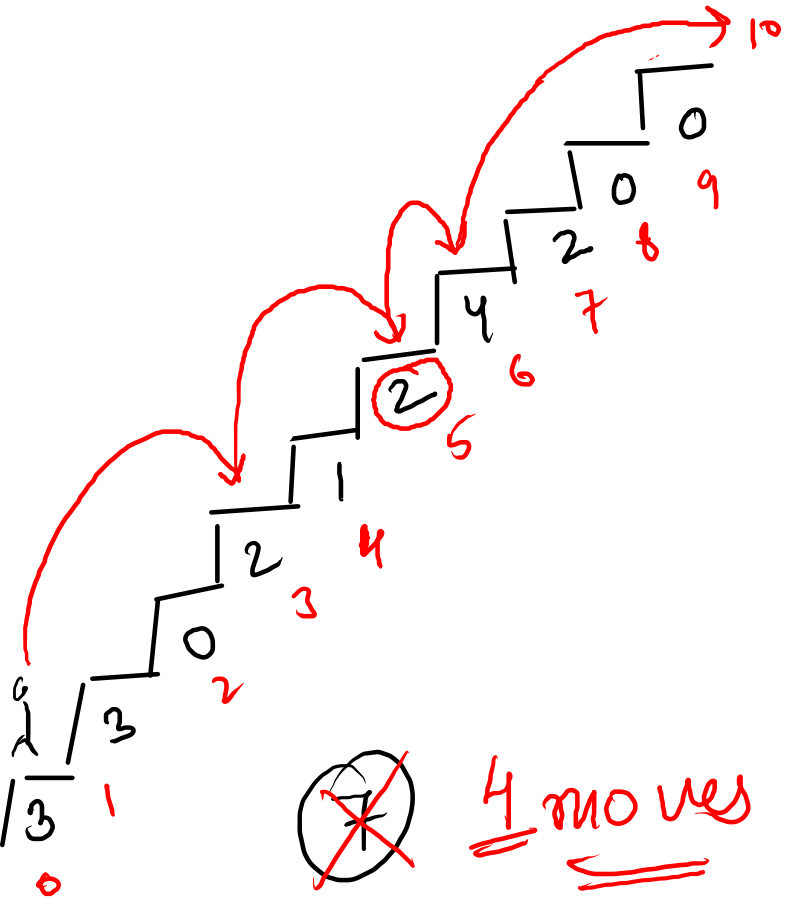
climb stairs with Variable Jump

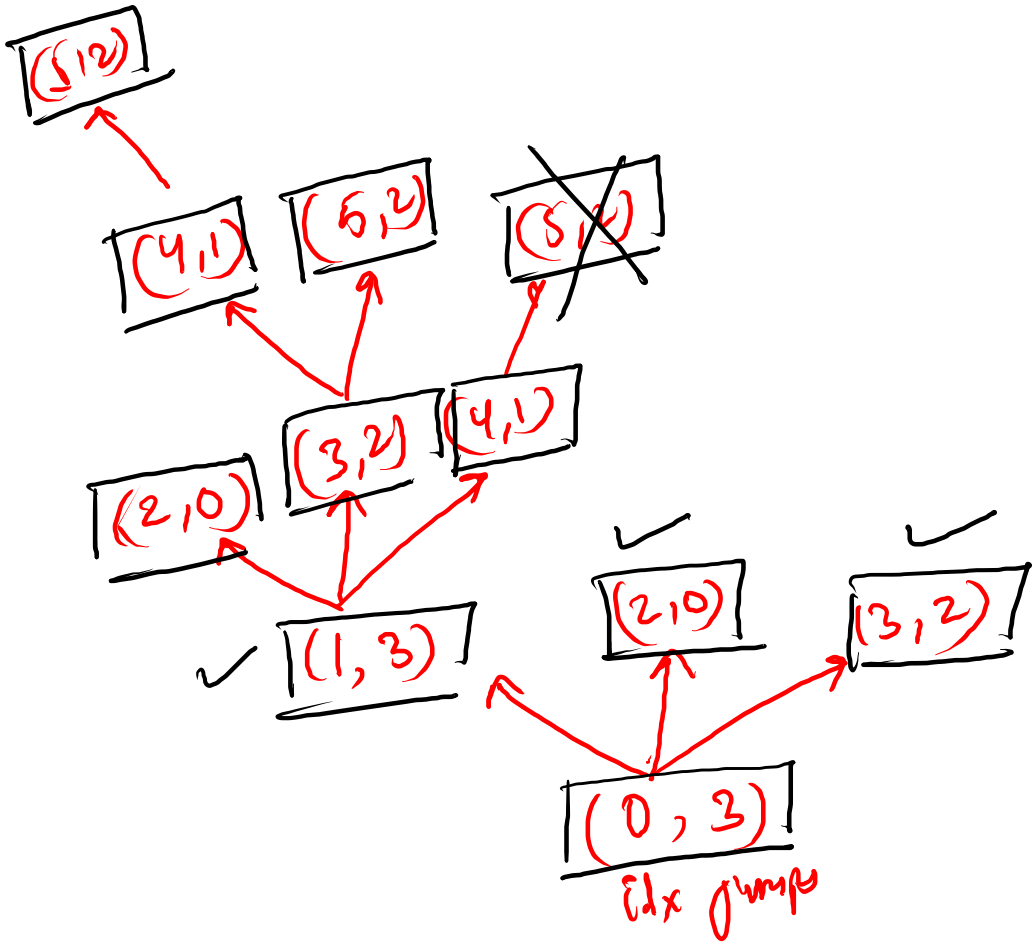


max length jump from that position

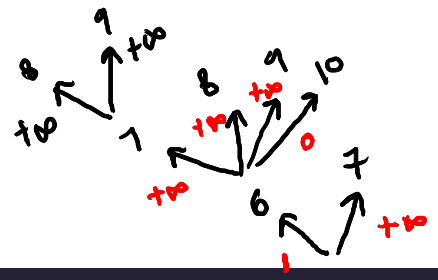


Climb stairs with minimum moves

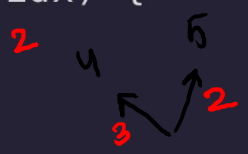


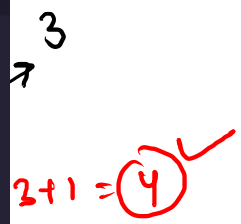


```
10
3
3
0
2
1
2
4
2
```



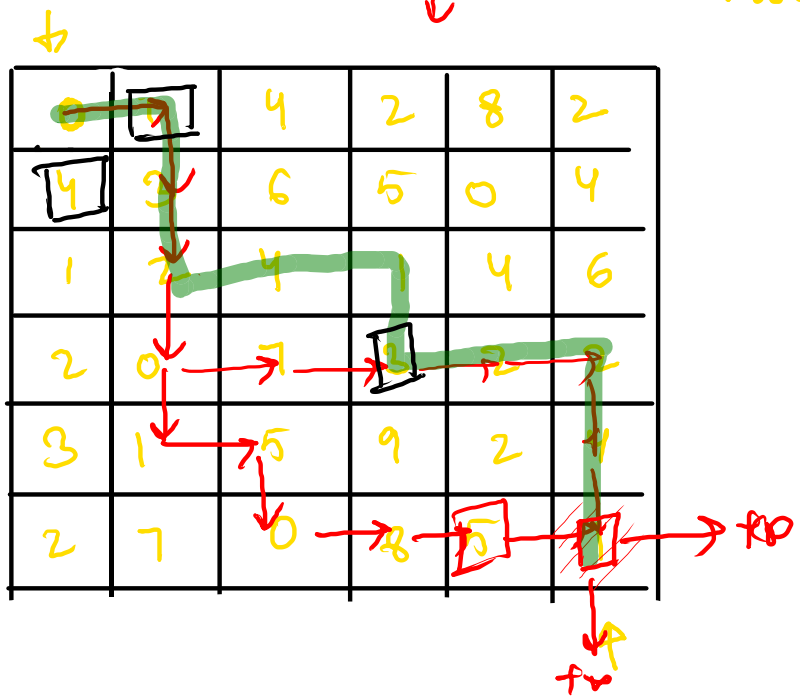
```java
public static int climb(int n, int[] jumps, int idx) {
    if(idx == n) {
        return 0;
    }

    int ans = Integer.MAX_VALUE;
    for(int jump = 1; jump <= jumps[idx]; jump++) {
        if(jump + idx <= n) {
            ans = Math.min(ans, climb(n, jumps, idx + jump));
        }
    }

    if(ans != Integer.MAX_VALUE) {
        ans += 1;
    }

    return ans;
}
```



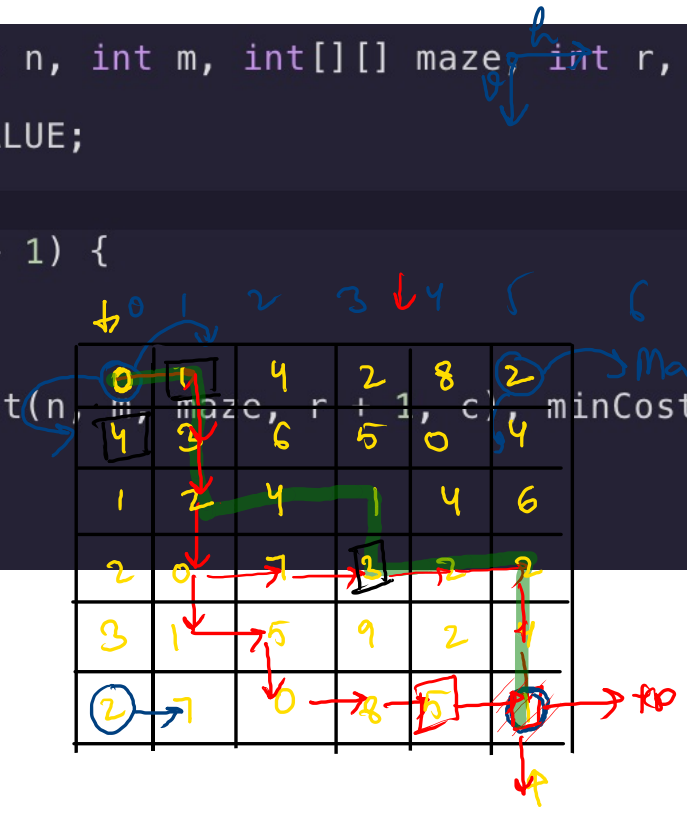


Minimum Cost in Maze traversal

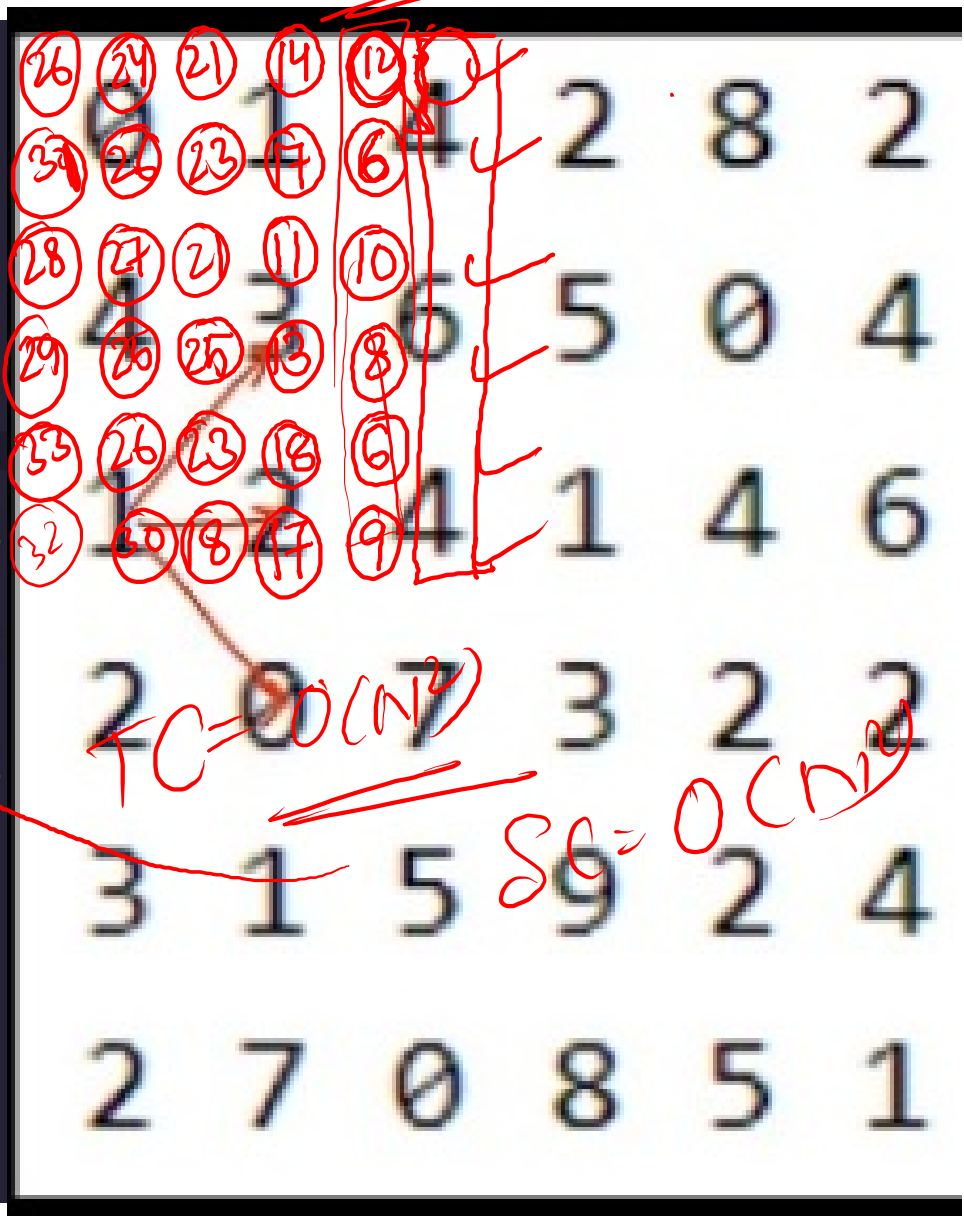
```
6
6
0 1 4 2 8 2
4 3 6 5 0 4
1 2 4 1 4 6
2 0 7 3 2 2
3 1 5 9 2 4
2 7 0 8 5 1
```



```java
public static int minCost(int n, int m, int[][] maze, int r, int c) {
    if(r == n || c == m) {
        return Integer.MAX_VALUE;
    }

    if(r == n - 1 && c == m - 1) {
        return maze[r][c];
    }

    int ans = Math.min(minCost(n, m, maze, r + 1, c), minCost(n, m, maze, r, c + 1));

    return ans + maze[r][c];
}
```



```java
// tabulation
int[][] dp = new int[n][m];
int sol  = Integer.MIN_VALUE;
for(int c = m - 1; c >= 0; c--) {
    for(int r = 0; r < n; r++) {
        if(c == m - 1) {
            dp[r][c] = mine[r][c];
            continue;
        }

        int ans = 0;
        if(r + 1 < n) {
            ans = Math.max(ans, dp[r + 1
        }

        ans = Math.max(ans, dp[r][c + 1]

        if(r - 1 >= 0) {
            ans = Math.max(ans, dp[r - 1
        }

        dp[r][c] = (ans + mine[r][c]);
    }
}

for(int r = 0; r <= n - 1; r++) {
    sol = Math.max(sol, dp[r][0]);
}
System.out.println(sol);
```

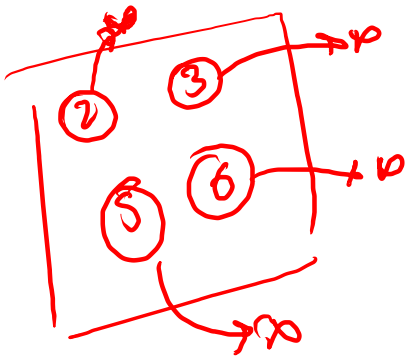


Coin Change Combination

```
4
2
3
5
6
7
```

Sample Output

```
2
```



target
7

{2,2,3}

{2,5}

Combination Vs permutation

{1,2}

{1,2}
{2,1}

coins = [2, 3, 5]

target = 15

{2, 2, 2, 2, 2, 2, 3}

{2, 2, 2, 2, 2, 5}

{2, 2, 3, 3, 5}

{2, 2, 2, 3, 3, 3}

{5, 5, 5}

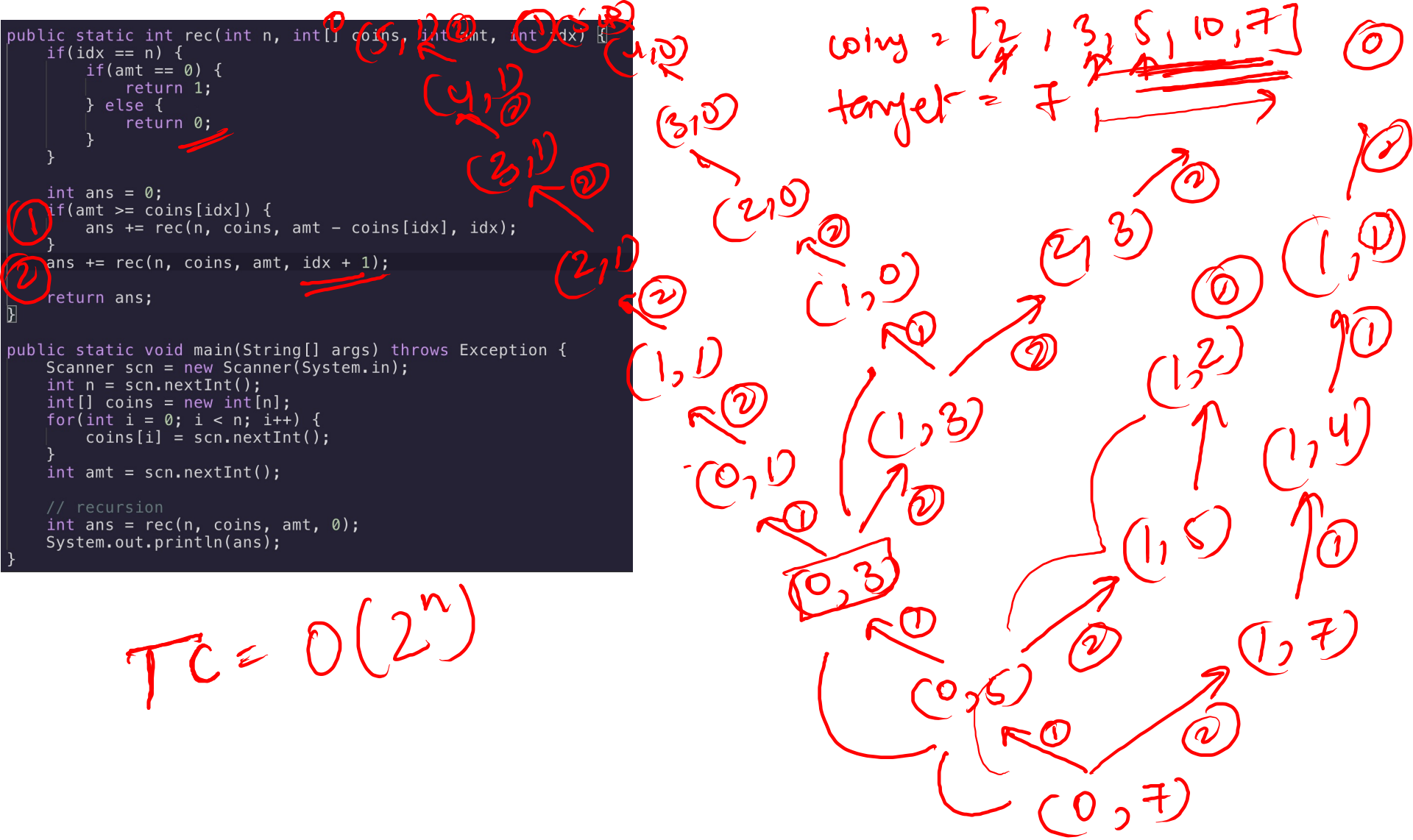
{3, 3, 3, 3, 3}

{2, 3, 5, 5}

coins = [2, 3, 5]

target = 5

```java
public static int rec(int n, int[] coins, int amt, int idx) {
    if(idx == n) {
        if(amt == 0) {
            return 1;
        } else {
            return 0;
        }
    }

    int ans = 0;
    if(amt >= coins[idx]) {
        ans += rec(n, coins, amt - coins[idx], idx);
    }
    ans += rec(n, coins, amt, idx + 1);

    return ans;
}

public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] coins = new int[n];
    for(int i = 0; i < n; i++) {
        coins[i] = scn.nextInt();
    }
    int amt = scn.nextInt();

    // recursion
    int ans = rec(n, coins, amt, 0);
    System.out.println(ans);
}
```

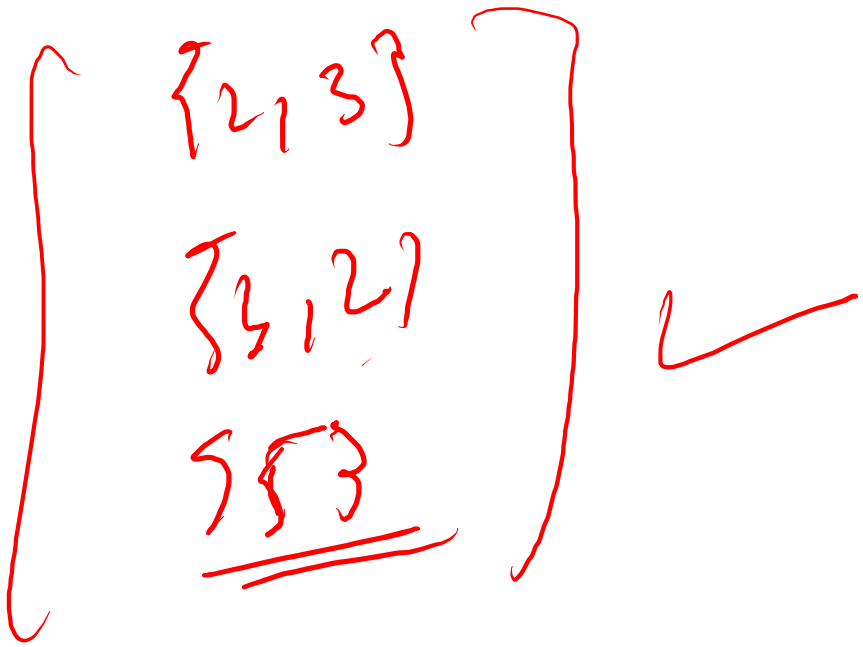


$TC = O(2^n)$

```java
// tabulation
int[][] dp = new int[n + 1][amt + 1];
for(int idx = n; idx >= 0; idx--) {
    for(int a = 0; a <= amt; a++) {
        if(idx == n) {
            if(a == 0) {
                dp[idx][a] = 1;
                continue;
            } else {
                dp[idx][a] = 0;
                continue;
            }
        }

        int ans = 0;
        if(a >= coins[idx]) {
            ans += dp[idx][a - coins[idx]];
        }
        ans += dp[idx + 1][a];

        dp[idx][a] = ans;
    }
}
System.out.println(dp[0][amt]);
```

```java
public static int memo(int n, int[] coins, int amt, int idx, int[][] dp) {
    if(idx == n) {
        if(amt == 0) {
            return dp[idx][amt] = 1;
        } else {
            return dp[idx][amt] = 0;
        }
    }

    if(dp[idx][amt] != 0) {
        return dp[idx][amt];
    }

    int ans = 0;
    if(amt >= coins[idx]) {
        ans += memo(n, coins, amt - coins[idx], idx, dp);
    }
    ans += memo(n, coins, amt, idx + 1, dp);

    return dp[idx][amt] = ans;
}
```

[2,3,5] target = 5

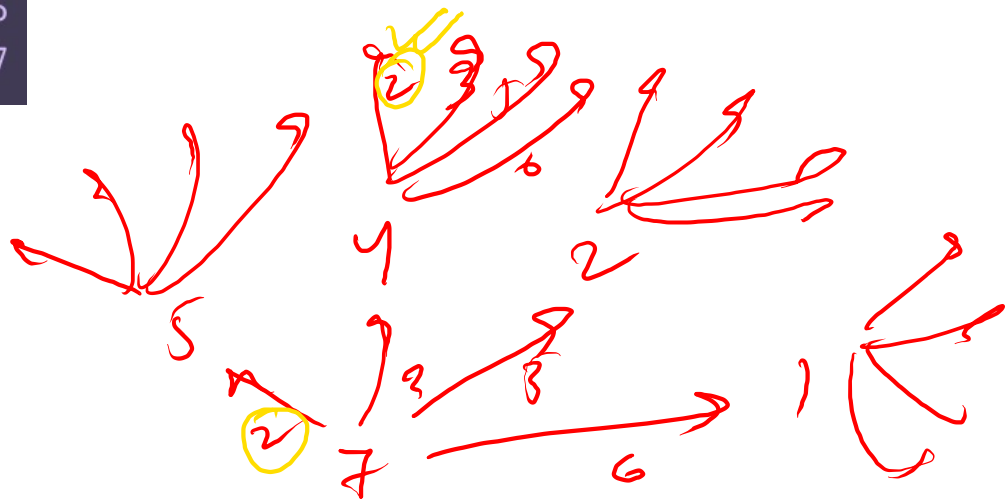


```java
public static int rec(int n, int[] coins, int amt) {
    if(amt == 0) {
        return 1;
    }

    int ans = 0;
    for(int coin : coins) {
        if(amt - coin >= 0) {
            ans += rec(n, coins, amt - coin);
        }
    }

    return ans;
}
```

```
4
2
3
5
6
7
```

coins = [2, 3, 5, 6]

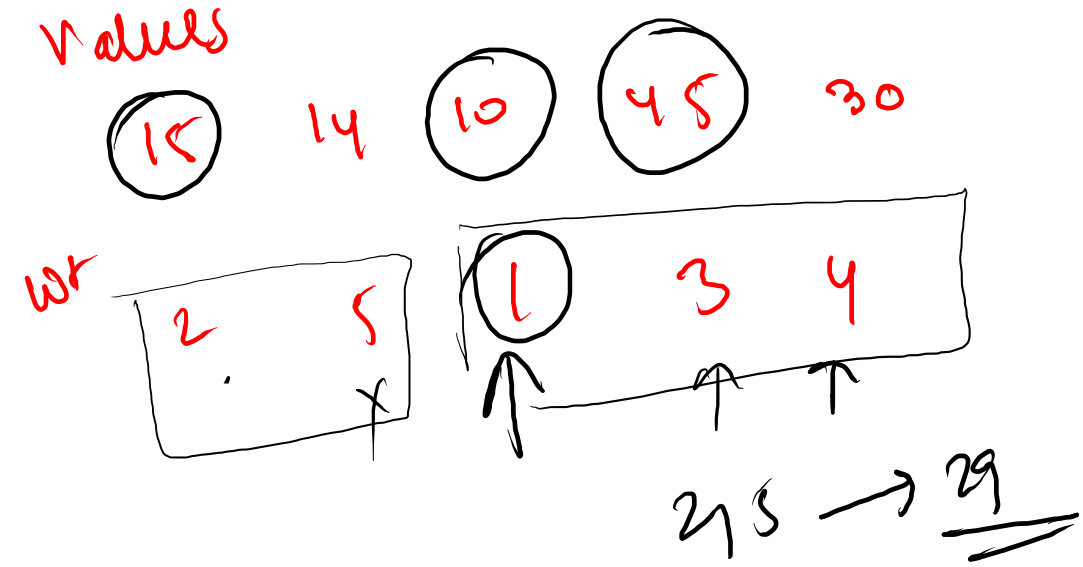
amt = 7

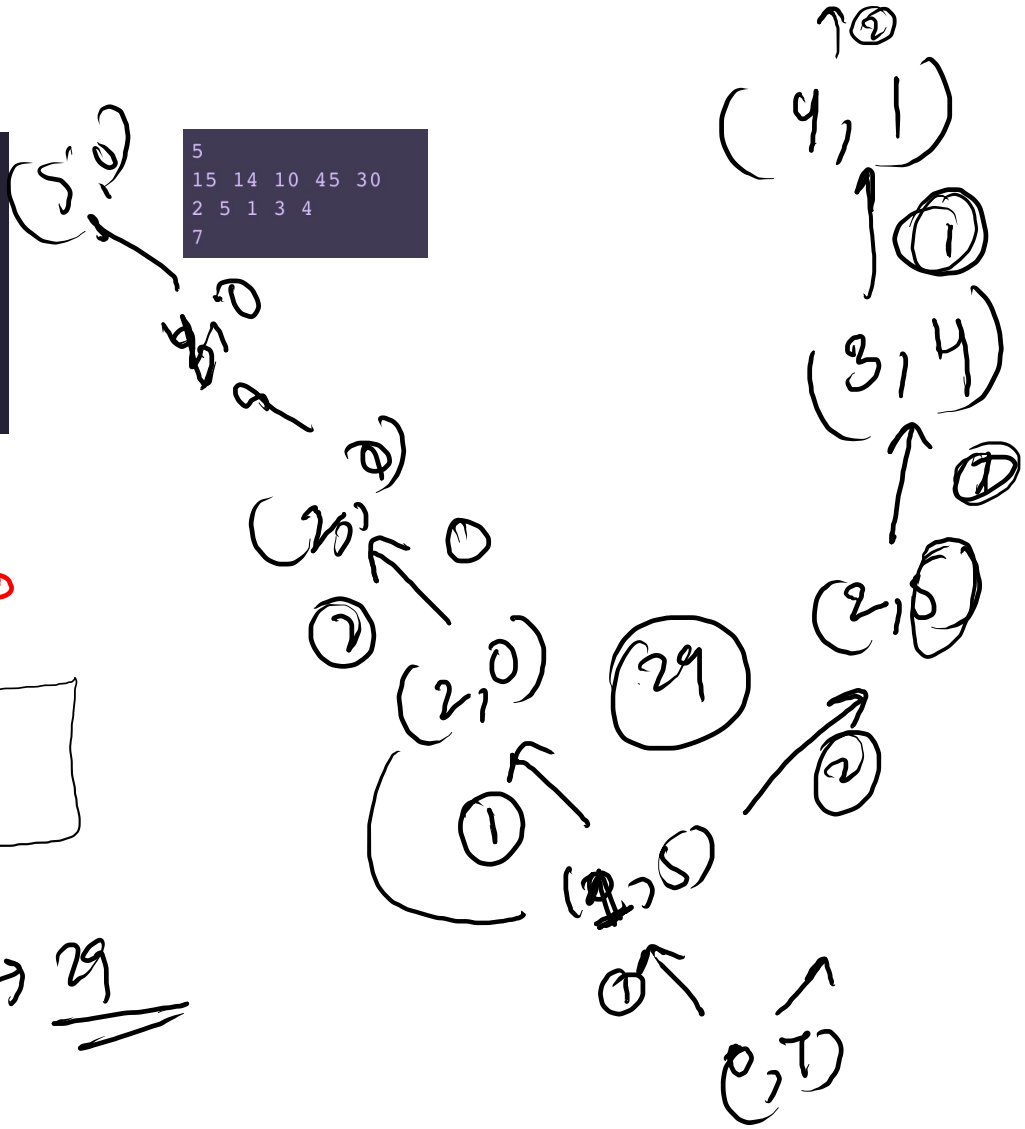


0 1 Knapsack

```java
public static int rec(int n, int[] val, int[] wt, int cap, int idx) {
    if(idx == n) {
        return 0;
    }

    int inc = 0, exc = 0;
    if(cap >= wt[idx]) {
        inc = rec(n, val, wt, cap - wt[idx], idx + 1) + val[idx];
    }
    exc = rec(n, val, wt, cap, idx + 1);

    return Math.max(inc, exc);
}
```

```
5
15 14 10 45 30
2 5 1 3 4
7
```





```java
    // System.out.println(ans);

    // tabulation
    int[][] dp = new int[n + 1][cap + 1];
    for(int idx = n; idx >= 0; idx--) {
        for(int c = 0; c <= cap; c++) {
            if(idx == n || c == 0) {
                dp[idx][c] = 0;
                continue;
            }

            int inc = 0, exc = 0;
            if(c >= wt[idx]) {
                inc = dp[idx + 1][c - wt[idx]] + val[idx];
            }
            exc = dp[idx + 1][c];

            dp[idx][c] = Math.max(inc, exc);
        }
    }
    System.out.println(dp[0][cap]);
}
```