current PSP 55.07 → 60

Nov23_PSP_19Apr
Gobika K
Piyush Kumar
kameswarreddy Yeddula
Manjunatha I
Kevin Theodore E
Harshil Dabhoya
Sai Sharath
Mohammad Mateen
Vijay V A
Rajeev
Yash Malviya
Suraj Devraye
Robin Dhiman
manikandan m

Nov23_PSP_19Apr
Mayur Hadawale
SABBAVARAPU KARTHIK
MD JASHIMUDDIN
Nitendra Rajput
Vigneshwaran K
Prashant Kumar Soni
Sarat Patel
Pradeep Kumar Chandra
Shaurya Srivastava
Mohammed Arshad
sudhakar venkatachalam
Pranadarth S
SIJU SAMSON
Pushkar Deshpande

Contest → Trees, Heaps, Greedy

Doubts Session → Sunday (Assignments)

Q → Introduction to DP using Fibonacci

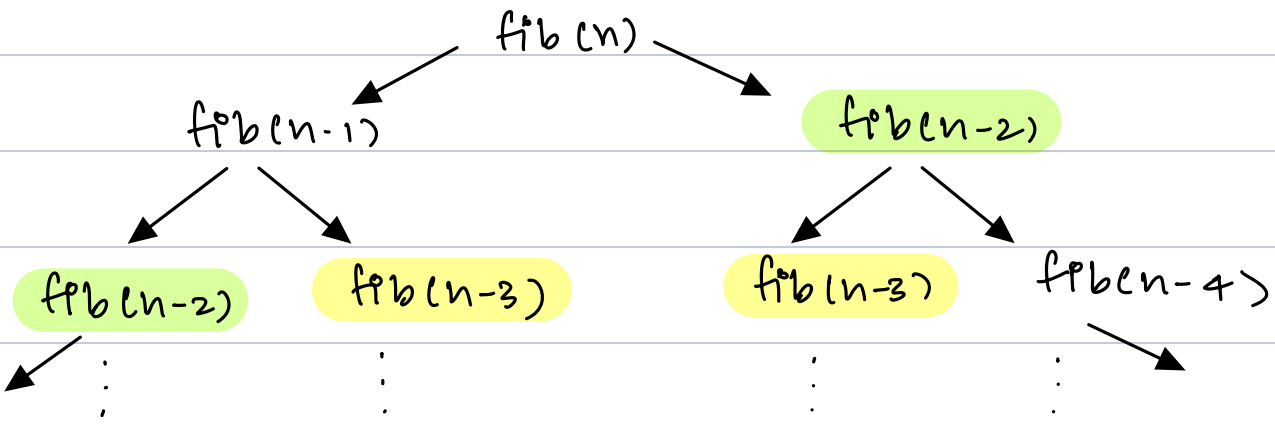
fib =

0	1	1	2	3	5	8	13	21	...

pseudo code

```
int fib(n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

Recursion call



T.C = O(2^n) S.C = O(n)

How to identify if we can solve the problem using DP?

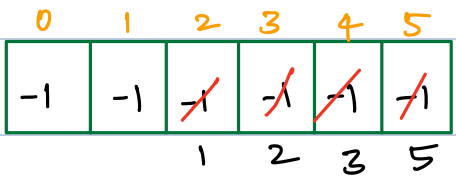
1. Optimal Substructure → can split the given problem into various subproblems
2. Overlapping subproblems → Repeating subproblem

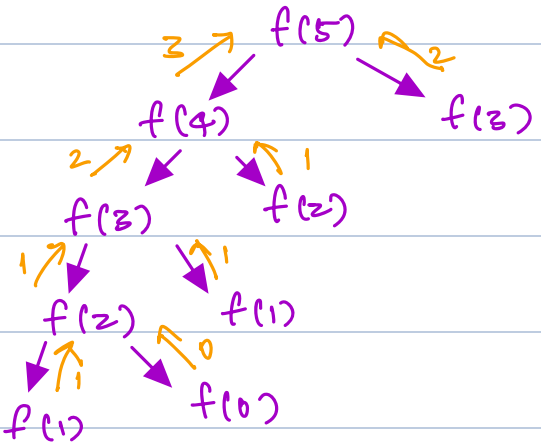
How to solve?

Store the ans of subproblem & reuse

Dynamic Programming solution

```
int F[N+1]; //Initialize to -1;
int fib(n) {
    if (n <=1) return n;
    if (F[N] != -1) return F[N];
    F[N] = fib(n-1) + fib(n-2);
    return F[N];
}
```





T.C = O(n) S.C = O(n)

Types of Dynamic Programming solution

[easy to implement]

Top Down Approach : [Memoization]

a) It is a recursive solution

b) start with the biggest problem and keep on breaking it till we reach base case

c) store the answer of already evaluated problems and reuse it

Bottom up Approach : [saves potential space]

1) It is an iterative solution

2) We start with the smallest problem, solve and store its result

3) Keep moving to bigger problem using already calculated solution

Fibonacci using Bottom up Approach

```
F[0] = 0    F[1] = 1
for i = 2 to n {
    F[i] = F[i-1] + F[i-2];
}
return F[n]
```

T.C = O(n)

S.C = O(n)

Optimizing Space

```
first = 0, second = 1;
for i = 2 to n {
    third = first + second;
    first = second;
    second = third;
}
return third;
```

T.C = O(n)
S.C = O(1)

Quiz 1

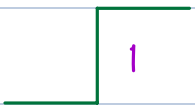
Purpose of memoization is to store and reuse already computed subproblems.

Quiz 2

Iterative approach is also called Bottom Up

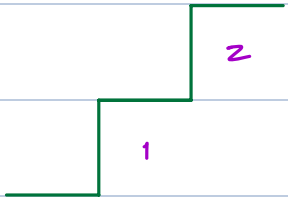
Q → Calculate the number of ways to reach the nth stair. You can take 1 step or you can take 2 steps at a time.

Best way to understand a problem is using examples



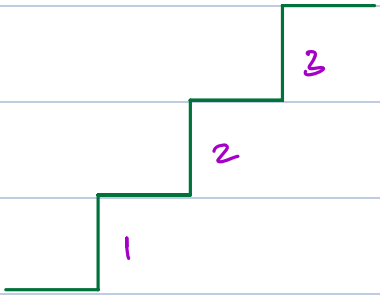
N = 1

options : [1]



N = 2

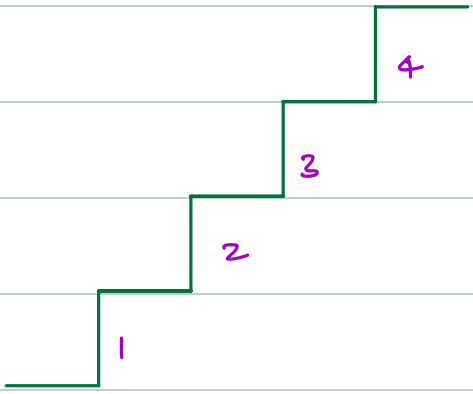
[1,1] [2] ⟶ 2



N = 3

options : [1,1,1] [1,2]

[2,1] ⟶ 3



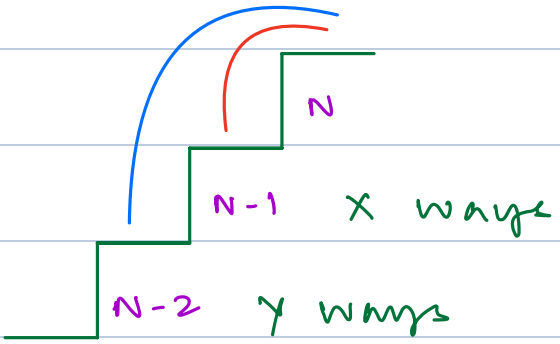
N = 4 (Quiz)

[1,1,1,1] [1,1,2] [1,2,1]

[2,2] [2,1,1]

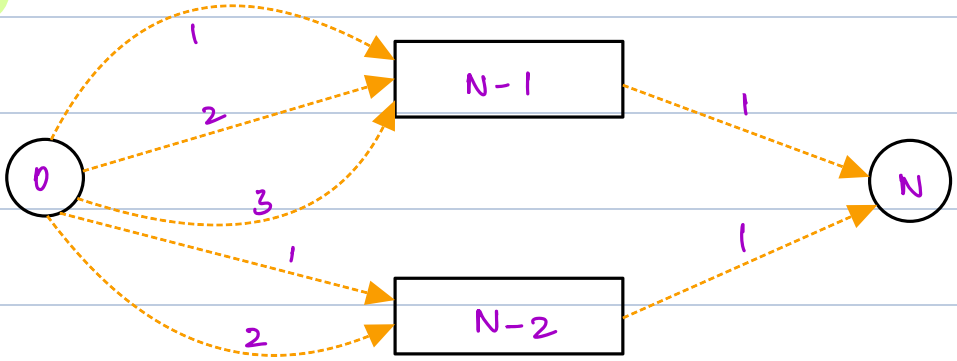
Last step approach

From where all can we reach N in one step ?



x + y steps

Visualize



ways (N) = ways (N-1) + ways (N-2)

exactly similar to fibonacci

Base conditions

N = 1 return 1

N = 2 return 2

T.C = O(n) S.C = O(1)

Break : 10:16 pm

Q → Find minimum number of perfect squares required to get sum = N (duplicate squares are allowed)

N = 6 → $1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$ [6]

$2^2 + 1^2 + 1^2$ [3]

N = 10 → $1^2 + 1^2 + 1^2 + \ldots + 1^2$ [10]

$2^2 + 2^2 + 1^2 + 1^2$ [4]

$3^2 + 1^2$ [2]

Quiz 4

N = 5 ⟶ $1^2+1^2+1^2+1^2+1^2$ [5]

2^2+1^2 [2]

Greedy Approach

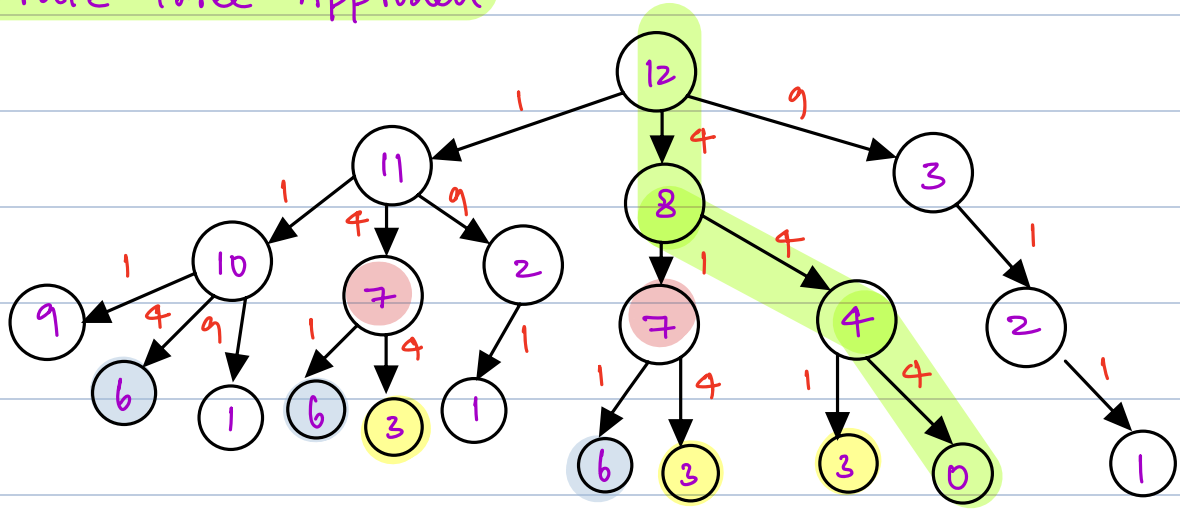
Subtract highest possible perfect square from N

example N = 12

$12 = 3^2+1^2+1^2+1^2$ [4] ⟶ greedy does not work

$12 = 2^2+2^2+2^2$ [3]

Brute force Approach



Recurrence Relation

$$F(12) = \min \begin{cases} 1+F(11) \\ 1+F(8) \\ 1+F(3) \end{cases}$$

solve (x) = 1 + min (solve (x - i²)) ∀i i * i <= x

pseudo code

```
Int psquare (Int n) {
    if (N==0) return 0;
    ans = N; // worst ans consider 1^2 only
    for (x=1; x*x <= n; x++) {
        ans = 1 + min (ans, psquare (n - x^2));
    }
    return ans;
}
```

Dynamic Programming

subproblem

Optimal substructure ⟶ Bigger problem to smaller ^

Overlapping subproblems ⟶ repeated subproblems

pseudo code

```
DP [n+1] // Initialize all 0 as -1
Int psquare (Int n) {
    if (N==0) return 0;
    if (DP [n] != -1) return DP [n]; // reuse
    ans = N; // worst ans consider 1^2 only
    for (x=1; x*x <= n; x++) {
        ans = 1 + min (ans, psquare (n - x^2));
```

```
    }
    DP[N] = ans; // store
    return ans;
}
```

T.C = O(n √n)

S.C = O(n)

// Iterative

```
DP[n+1]  // Initialize ∞
DP[0] = 0      DP[1] = 1
for (i = 2; i <= n; i++) {
    ans = N; // worst ans consider 1^2 only
    for (x = 1; x*x <= i; x++) {
        DP[i] = min( DP[i], 1 + DP[i - x^2]);
    }
}
return DP[n]
```

Q → RBI wants to reduce paper usage for money. Imagine you need to withdraw a specific amount of money from an ATM. The ATM should be programmed to give least number of notes.

Available notes : { 50, 30, 5 }

Return minimum number of notes for a given request.

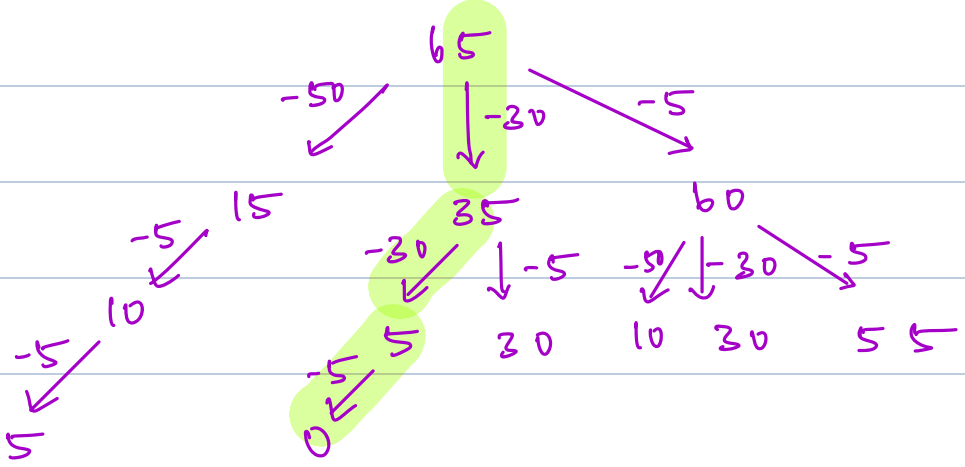
N = 100 → 50 * 2 = 100 (2)

N = 55 → 50 * 1 + 5 * 1 (2)

Greedy

N = 65 50 * 1 + 5 * 2 = (4 notes)

30 * 2 + 5 * 1 = (3 notes)



pseudo code Brute Force Store & reuse

```
DP[n-1] // Initialize to -1

int minNotes(int N) {
    if (N==0) return 0;
    if (N<0) return INT_MAX;
    // subtract 50, 30, 5
    if (DP[N] != -1) return DP[N];
    ans = INT_MAX;
    ans = min(1+minNotes(N-50), ans)
    ans = min(1+minNotes(N-30), ans)
    ans = min(1+minNotes(N-5), ans)
    DP[N] = ans;
    return DP[N]
}
```

T.C = O(n)