28/8/2023

DP-1

chocolates → [3 2 3 1 5] = Total → 14

$$\begin{array}{ccccc} 3 & 2 & 3 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{array}$$

1 new student → 2 chocolates

Total

A → 2+3+2+3+1+5 = 16

B → 2+14 = 16   save time & effort

↳ using already calculate value

⇒ dynamic programming.

Prefix sum

A = [3 2 3 1 5]

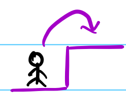
B = [3 5 8 9 14]

$P[i] = A[i] + P[i-1]$

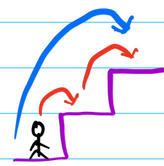
already calculated

Q In how many ways can we climb N stairs, s.t in one step we can climb 1 or 2 stairs only.

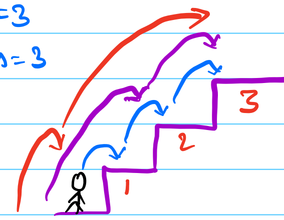
N=1
Ans=1



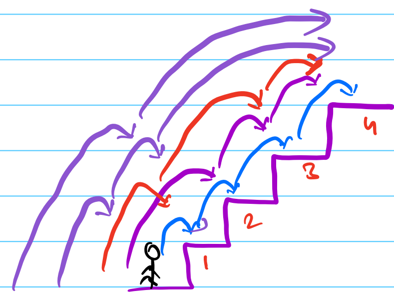
N=2
Ans=2



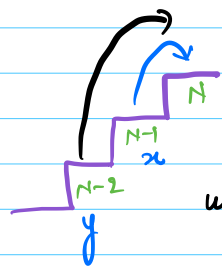
N=3
Ans=3



N=4
Ans=



1 1 1 1
2 1 1
1 2 1
1 1 2
2 2



$\boxed{\text{Last step}}$

$x + y$.

ways(4) = way(3) OR + way(2)

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

N=0
Ans=1 (no move)

\# ways = 0 ⇒ impossible task
not doing anything ⇒ 1 option.

	1	1	2	3	5	
N =	0	1	2	3	4	5 ......

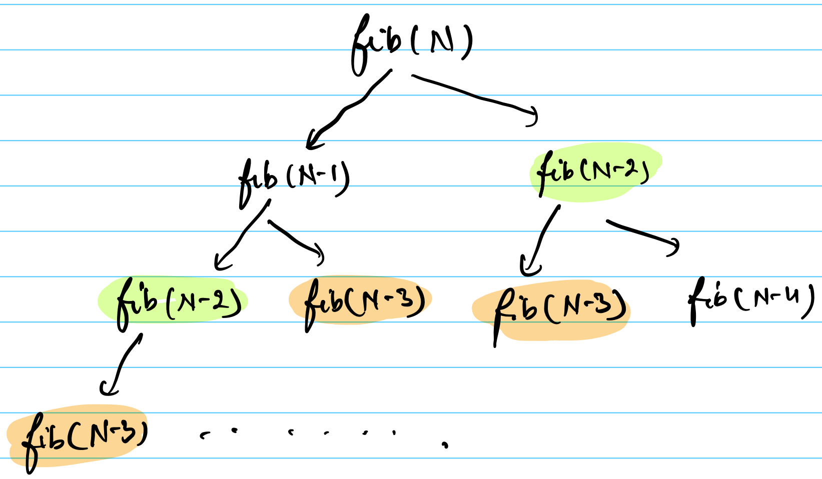
fibonacci numbers

	0	1	1	2	3	5	8	13	21	....
N =	0	1	2	3	4	5	6	7	8	....

Recursive code

TC: $O(2^n)$
SC: $O(N)$

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



1> Optimal substructure :- Ans of a big problem can be calculated using ans of its subproblems.

2> Overlapping subproblems :- Same subproblem is calculated multiple times.

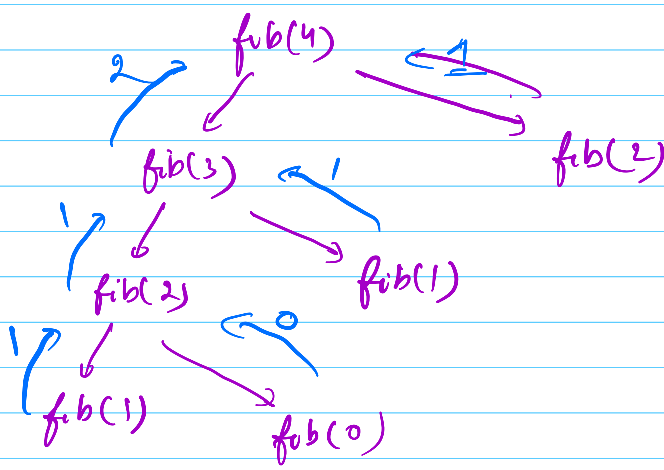
store the answer of subproblems to avoid recomputations

```
int fib(N){            int F[N]

   if (N <= 1) return N;

   if (F[N]>0) return F[N]

   F[N] = fib(N-1) + fib(N-2);

   return F[N];
}
```

TC: $O(N)$

SC: $O(N + N) \approx O(N)$

Recursion   F[N]



Top down / Recursive dp :- Start with big problem, go down till we reach smaller subproblems for which we already know the answer. use that to compute answer of current problem.

Bottom up / Iterative dp :- Start with smallest subproblem for which we already know the answer, use it to iteratively get the answer of current problem.

```
f[0] = 0 , f[1] = 1
for i-> 2 to N {
    f[i] = f[i-1] + f[i-2]
}
return f[N]
```

TC: O(N)
SC: O(N)

```
a = 0 , b = 1
for i-> 2 to N {
    c = a+b;
    a = b
    b = c
}
return c;
```

a b c
a b c

TC: O(N)
SC: O(1)

Recursive dp -> easy to write code.

Iterative dp -> No recursive space :- There are chances to optimize space.

Meet at 8:30 am

Q. Find the min no of perfect squares to add to get Sum = N, 1, 4, 9, 16, 25 . . . N>0.

N		Ans
1	1^2	1
2	1^2+1^2	2
3	$1^2+1^2+1^2$	3
4	2^2	1
5	1^2+2^2	2
.	.	.
10	1^2+3^2	2

x − (largest perfect square <= x) → greedy solution

N = 80

$80-8^2 = 80-64 = 16$

$16-4^2 = 16-16 = 0$

Ans = 2

N = 12

$12-3^2 = 12-9 = 3$

$3-1^2 = 3-1 = 2$

$2-1^2 = 2-1 = 1$

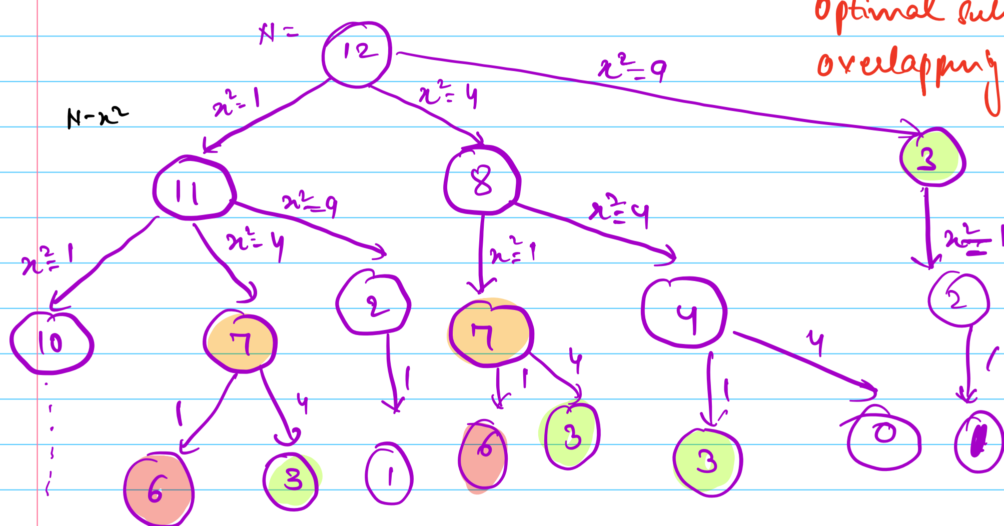
$1-1^2 = 1-1 = 0$

Ans = 4

$12 = \underbrace{2^2+2^2+2^2}{} = 4+4+4 = 12$

Ans = 3

Optimal subst ✓
overlapping sub ✓
DP



$count[N] = min(1 + count[N - x^2]$

$\forall x$ s.t $(x^2 <= N)$

ans[0] = 0

```
for i -> 1 to N {
    ans[i] = i        // 1^2+1^2+1^2.... i times

    for(x=1 ; x*x <= i ; x++ ){

        ans[i] = min( ans[i], 1+ans[i-x*x]))
    }
}
return ans[N]
```

TC: O(N * sqrt(N))
SC: O(N)

N = 6

i →	0	1	2	3	4	5	6	....
Ans →	0	1	2	3	1	2	.....	

Doubt

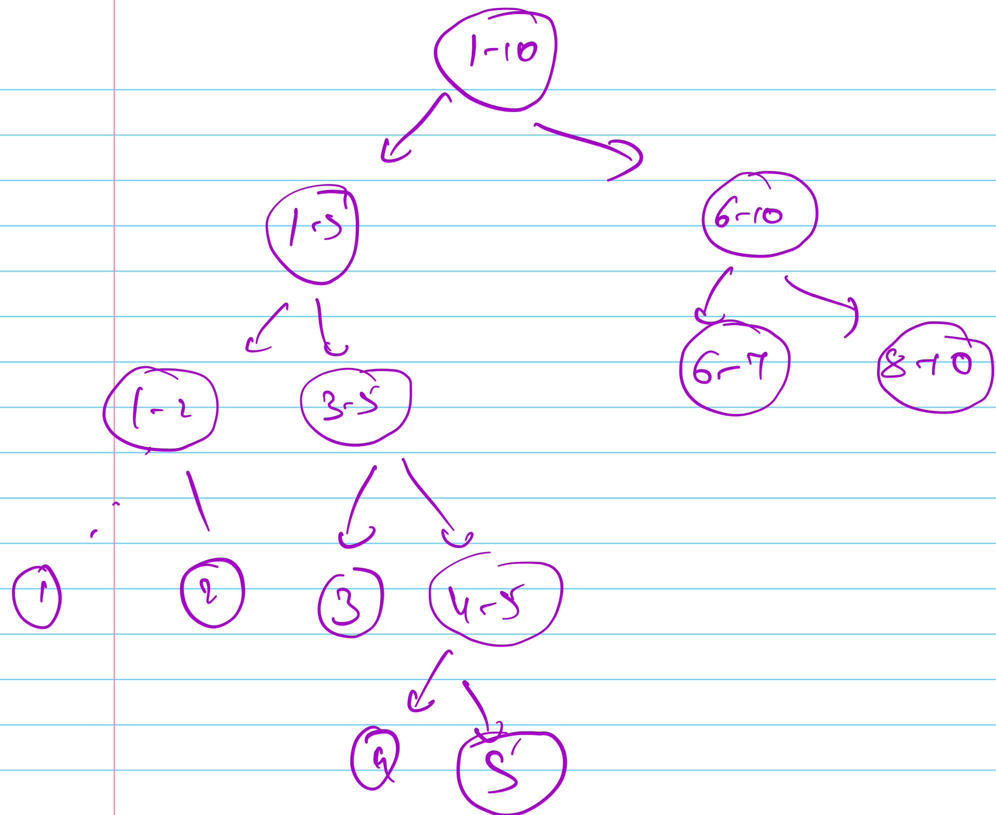
1 to n → 10^9 Q → 10^5 2 5
5, 7

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

1 2 . . . n

1, 5, , , 2 ,

7 . . — — 5


1-10
1-5
6-10
1-2
3-5
6-7
8-10
1
2
3
4-5
4
5