DP- 1

① Fibonacci No's

1	2	3	4	5	6	7	N	9	———
0	1	1	2	3	5	8	13	21	———

Q Given N. find out N^{th} fib No!

R.R → $f(N) = f(N-1) + f(N-2)$

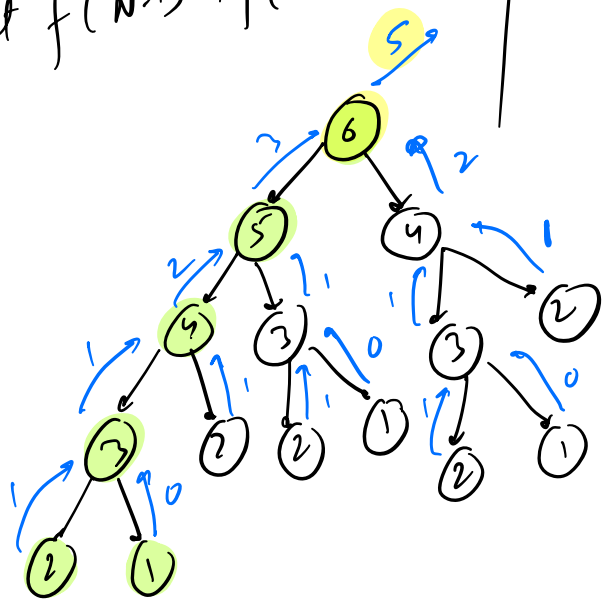
N >= 1

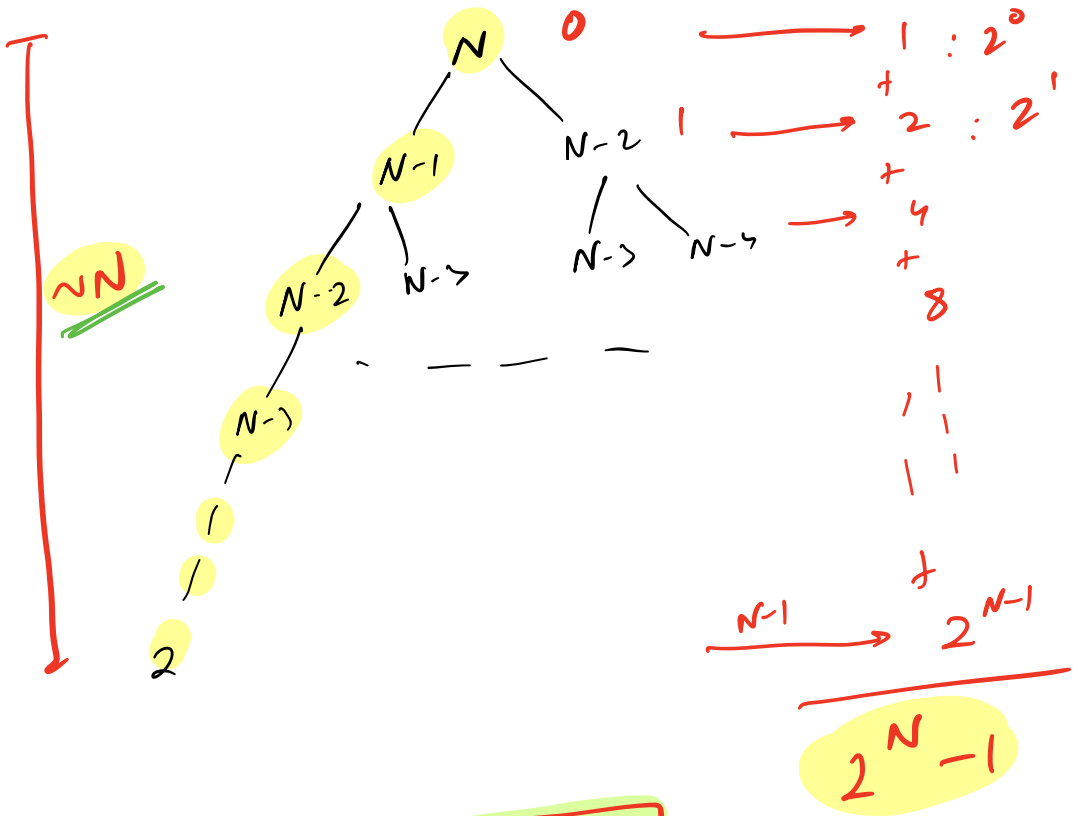
```
int f(N){
  if(N<=2) ret N-1;
  ret f(N-1) + f(N-2);
}
```

BC

N = 2 →

N = 1 →





f^n calls → O(2^N)

1 f^n call → O(1)

2^N f^n calls → O(2^N)

TC = O(2^N)

SC = O(N)

N = 40^th

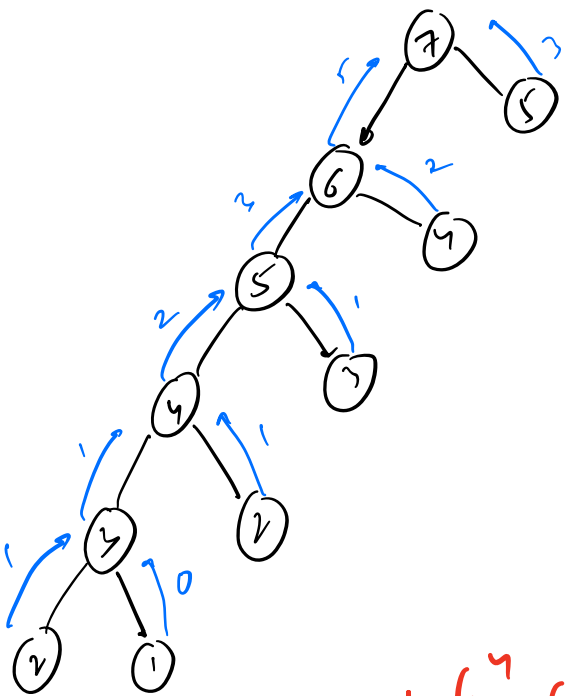
2^40

10^12

10^8 op → 1 sec

10^4 sec

Unique f^n calls → $O(N)$



HM

ANS

⟨3, 1⟩

⟨4, 2⟩

⟨5, 3⟩

⟨6, 5⟩

f^n calls → $\sim 2N$

$O(N)$

$O(1)/f^n$

TC = $O(N)$

D.P.

① We can solve a problem using DP when :-

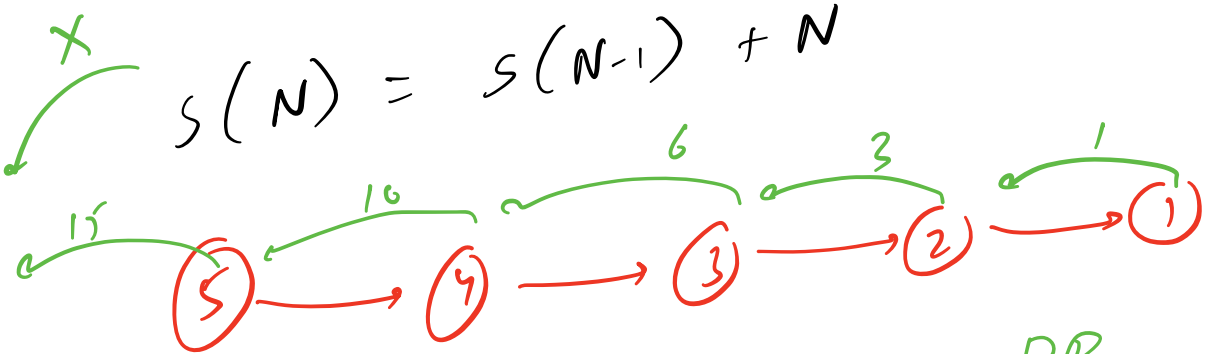
(1) Optimal substructure →

If the optimal ans of a problem can be obtained by optimal ans of subproblems!

(2) Overlapping subproblems →

→ Repitition of subproblems.

$$S(N) = S(N-1) + N$$



① Memoization → saving the results of the states (sub-problems)

Overlapping Subproblems

f^n calls > # Unique f^n calls

$2^N > N$

Recursion

```
int f(N){
  if(N<=2) ret N-1;
  ret f(N-1) + f(N-2);
}
```

TC = $O(2^N)$

DP

1) HM<int, int>
2) Array [] ✓

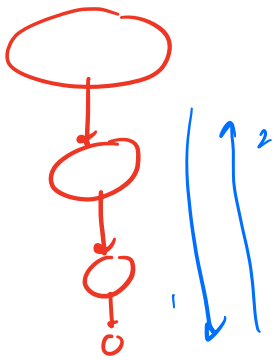
```
int dp[N+1] = {-1};
int f(x){
  if(x<=2) ret x-1;          // BC
  if(dp[x] != -1){           // Check
    ret dp[x];
  }
  ans = f(x-1) + f(x-2);     // Calculate
  dp[x] = ans;               // Store
  ret ans;                   // ret
}
```

TC = O(N) SC = O(N)

Two ways of solving DP problems

1) Top-down way

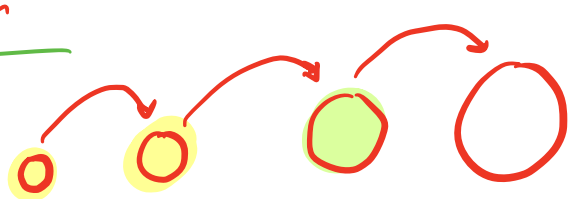
(Recursion + Memoization)



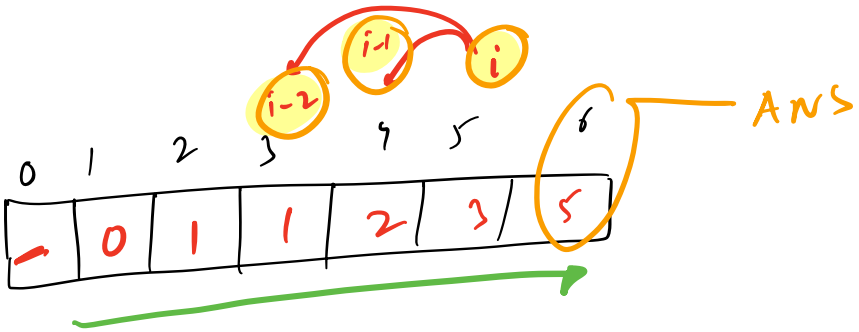
- → Easy to write/read
- → —— Understand
- → —— Come up with

2) Bottom Up Way

Iterative + Tabulation

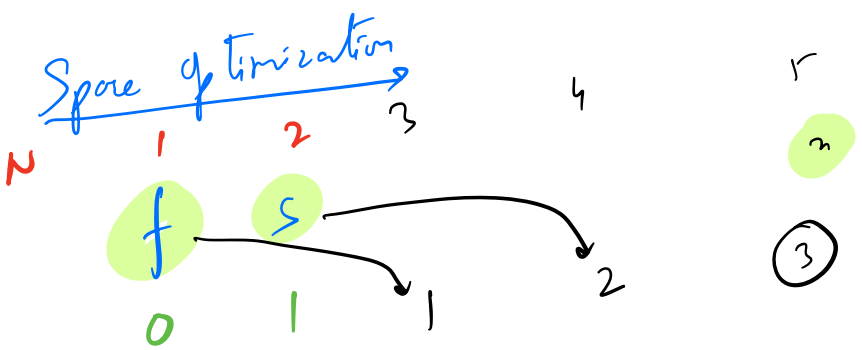


- → ✗ Write/Read
- → ✗ Come up with
- → ✓ Execution time less
- → ✓ Scope for space optimization
- → ✗ Author is responsible for deciding the order in which the ANS of states would be computed!




```
int dp[N+1];
dp[1] = 0, dp[2] = 1;
for( i = 3 ; i <= N; i++) {
        dp[i] = dp[i-1] + dp[i-2];
}
rt dp[N];
```

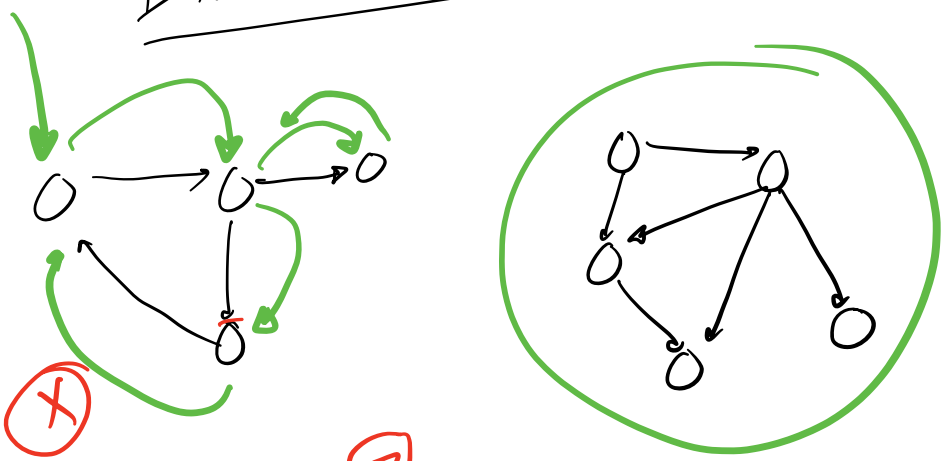
SC = O(N)

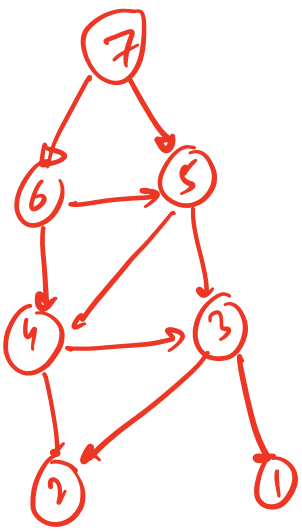


SC = O(1)

DAG

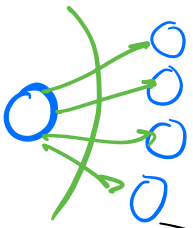
Directed Acyclic Graph





How ?

1) Identify the element of choice



2) How to represent a state ? → int

What will your state represent ? → as of the subproblem

define

3) Recurrence Relation

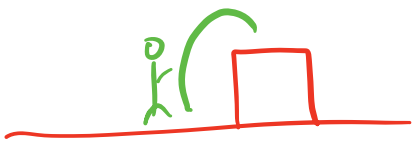
$ANS(n) = ANS(n-1) + A--(n-2)$

4) Which state contains your final ans !

→ f(N)

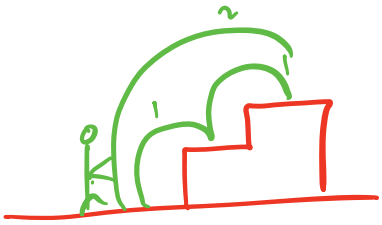
Q Given N steps in a stair.
Find the no. of ways of reaching N^{th} stair.
At a time, you can climb 1 stair or 2 stairs!

N=1



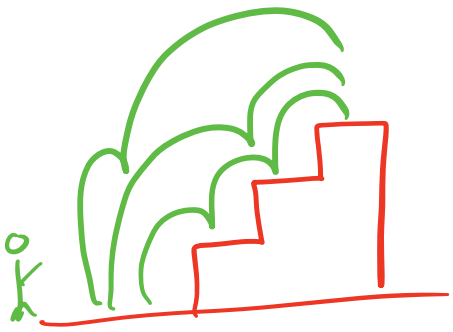
(1) → 1

N=2



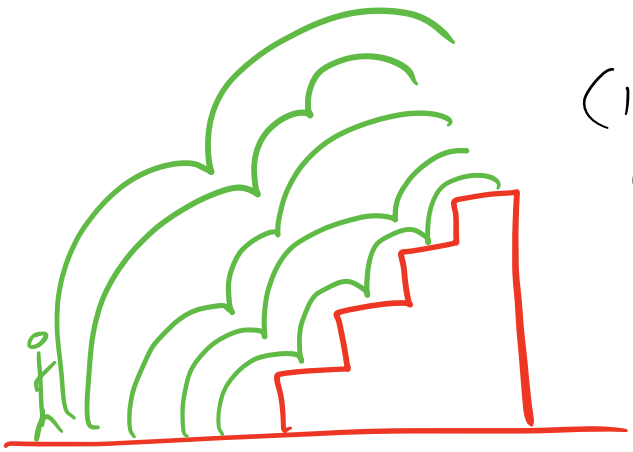
(1, 1)
(2) → 2

N=3



(1, 1, 1)
(1, 2)
(2, 1) → 3

N = 4



(1, 1, 1, 1)
(1, 2, 1) → 5
(1, 1, 2)
(2, 1, 1)
(2, 2)

⊙ Element of choice ?

no. of steps : 1, 2

(n)

⊙ How to represent a state ? → (int)

What does a state represent ?

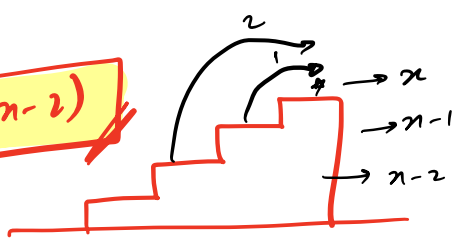
way(n) → the # of ways of reaching nth stair!

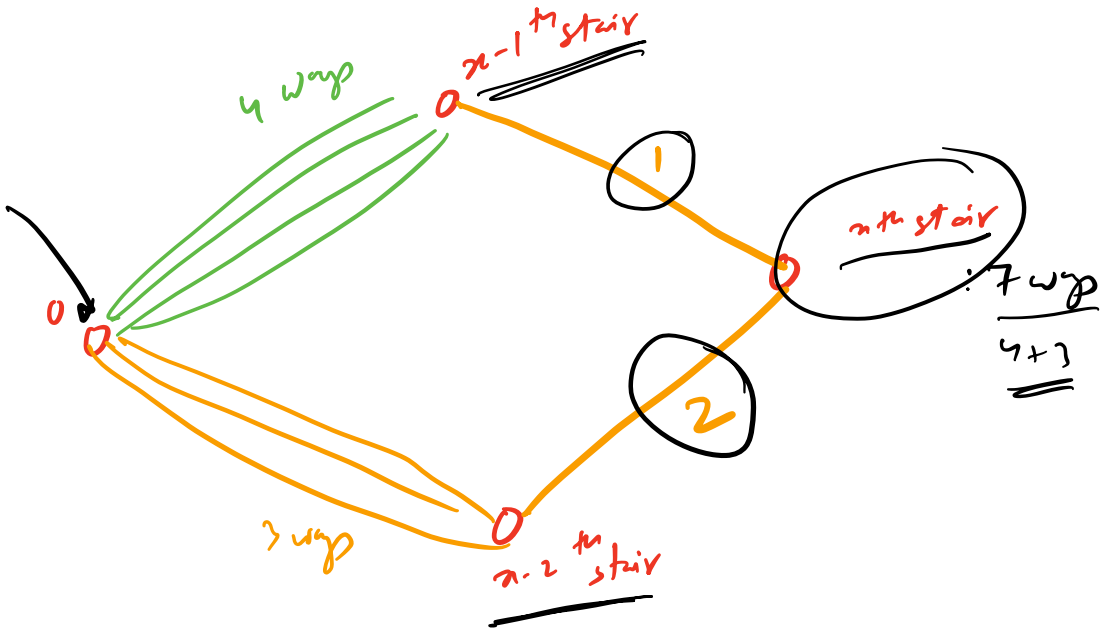
(n)

⊙ Recurrence Relation

way(n) = way(n-1) + way(n-2)

fibonacci

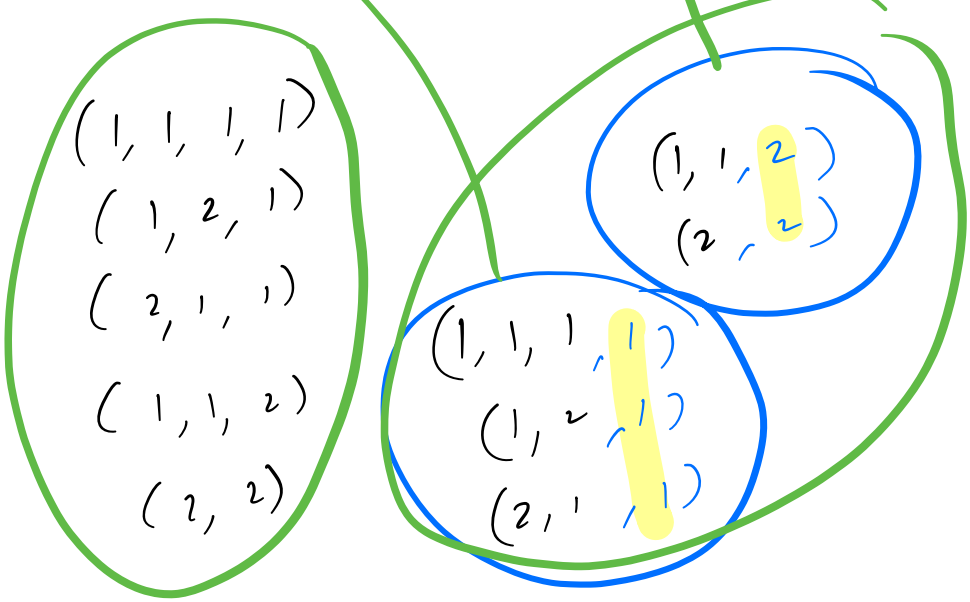


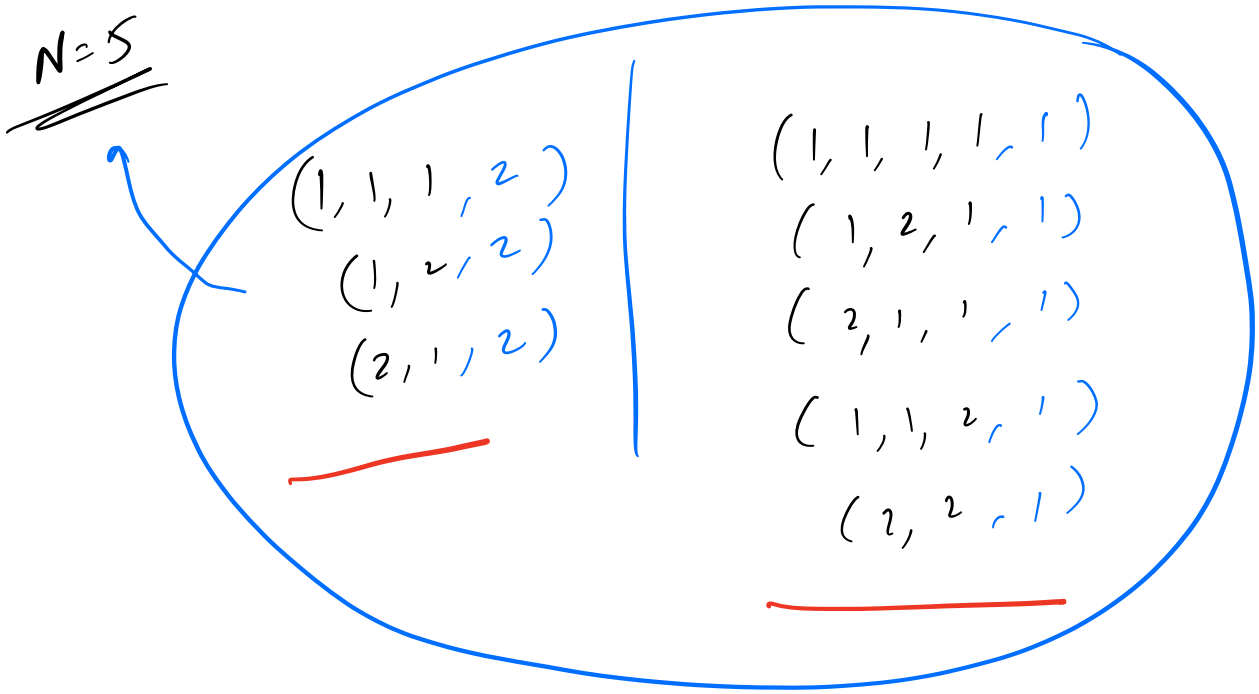


N=2 : (1, 1)
(2) : 2

N=3 : (1, 1, 1)
(1, 2)
(2, 1) : 3

N=4 :





① which state to find ans?

Ways(N)

ANS: Using fib - -

N=1 : 1

N=2 : 2

TC = O(N)

SC = O(N)

O(1)

Q Given N

Tell the min no. of no's whose sum of squares = N

$N = 100$: $10^2 \rightarrow 1$

$N = 101$: $10^2 + 1^2 \rightarrow 2$

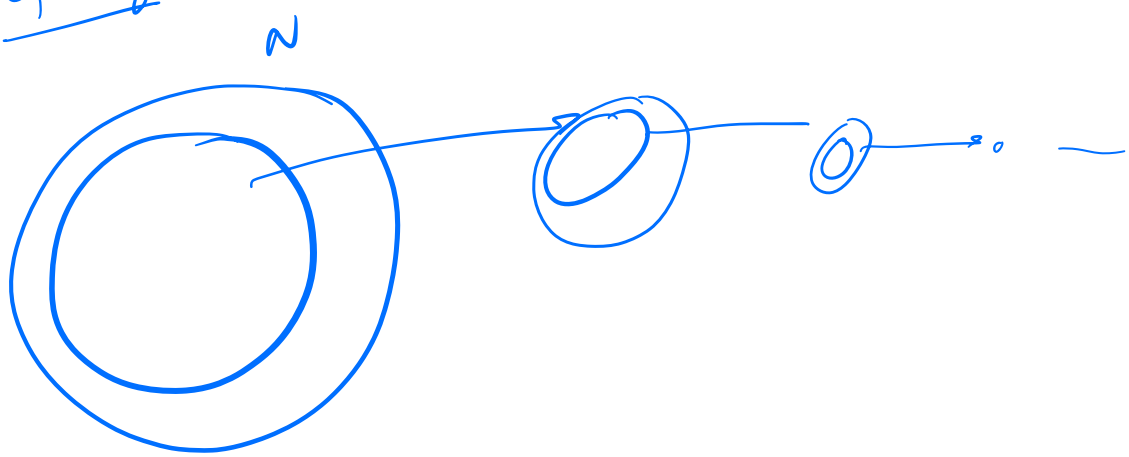
$N = 13$: $3^2 + 2^2 \rightarrow 2$

$N = 169$: $13^2 \rightarrow 1$

$N = 18$: $3^2 + 3^2 \rightarrow 2$

$N \rightarrow 1^2 + 1^2 + 1^2 + \dots + 1^2$: N

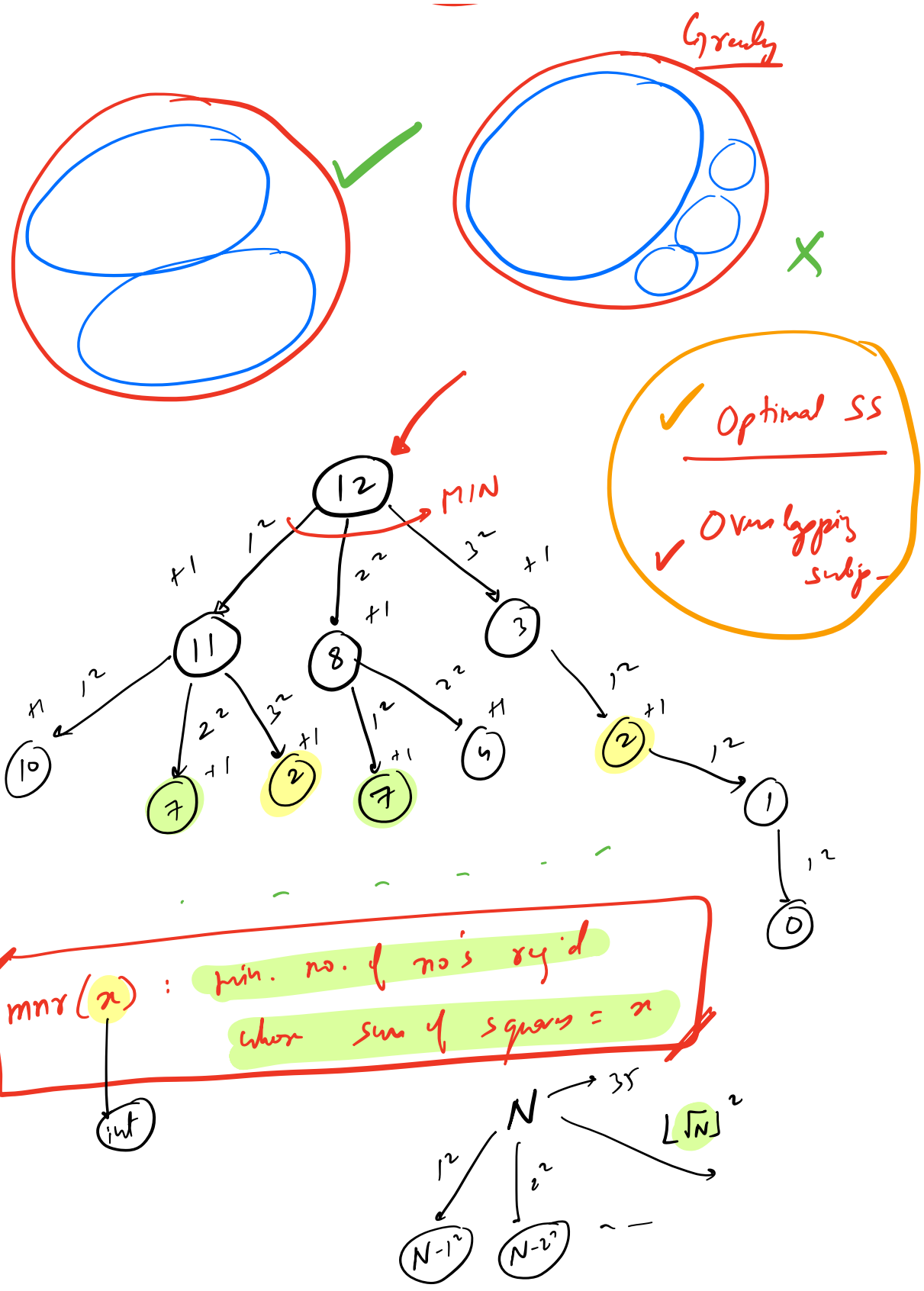
Greedy



$N = 18 \rightarrow 2 \rightarrow 1 \rightarrow 0$: 3 X

-4^2 -1^2 -1^2

$3^2 + 3^2$ → ANS → 2


Greedy
✓ Optimal SS
✓ Overlapping subp
MIN
12
11
8
3
10
7
2
7
4
2
1
0
mnr(x) : min. no. of no's req'd
whose sum of squares = x
int
N
$\lfloor\sqrt{N}\rfloor^2$
$N-1^2$
$N-2^2$

$$mnr(N) = 1 + \min \begin{cases} mnr(N-1^2) \\ mnr(N-2^2) \\ mnr(N-3^2) \\ \vdots \\ mnr(N-\lfloor\sqrt{N}\rfloor^2) \end{cases}$$

$$mnr(N) = 1 + \min_{K=1}^{K=\lfloor\sqrt{N}\rfloor} mnr(N-K^2)$$


```
int dp[N+1] = {-1};
int mnr(x) {
    if(x==0) ret 0;          // BC
    if(dp[x] != -1) {        // check
        ret dp[x];
    }
```

```
sq = sqrt(n);
ANS = 00;
for(k=1; k<=sq; k++) {
    ANS = min(ANS, mm(N - k*k));
}
ANS++;
dp[n] = ANS;
ret ANS;
}
```

Calculate

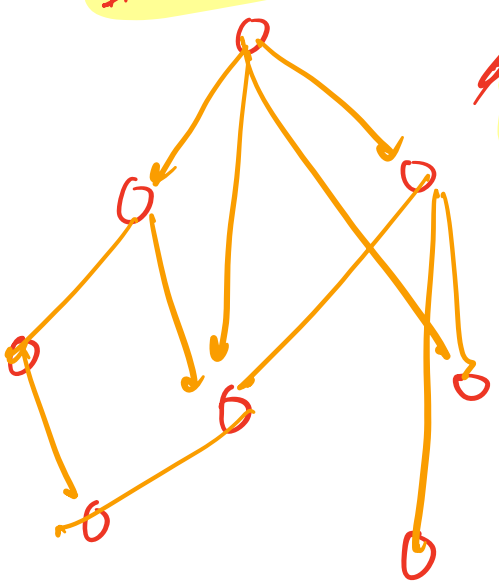
//store

// ret

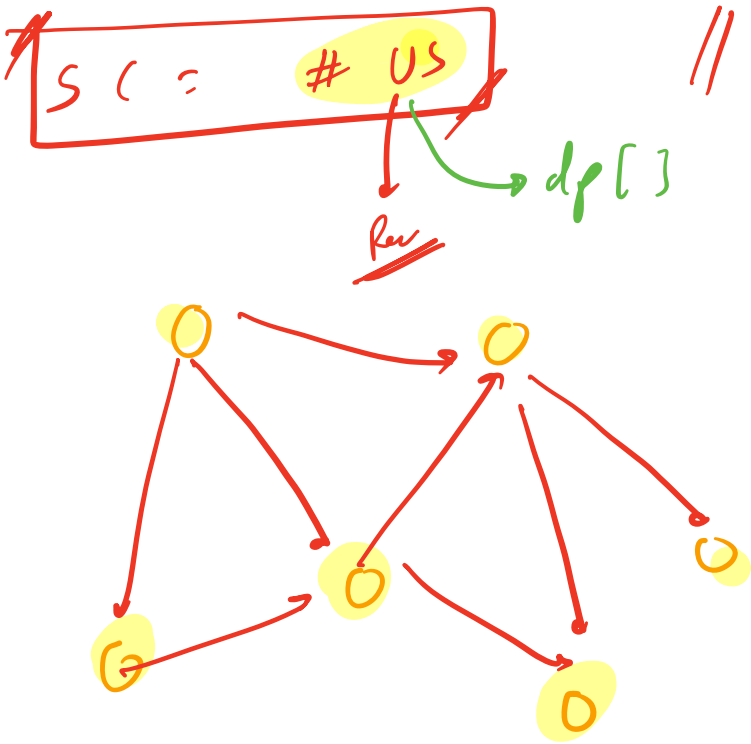
TC = # Unique states × time taken per state

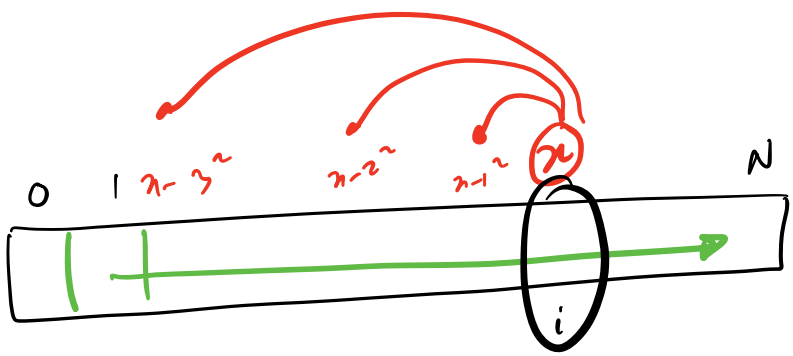
#US → ~N

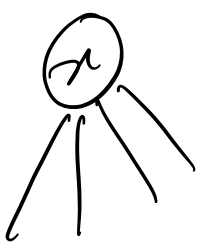
$\sqrt{N}$



TC = O(N√N)








```
int dp[N+1];
dp[0] = 0;

for(i=1; i<=N; i++){
    ANS = ∞;
    sq = sqrt(i);
    for(K=1; K<=sq; K++){
        ANS = min(ANS, dp[i - K*K]);
```

```
    }
    dp[i] = 1 + ANS;
}
return dp[N];
```

TC = O(N√N)

SC = O(N)