Today's agenda

- Dynamic Programming Intro
- When to use DP
- Steps for DP
- # N Stairs
- Sqrt



→ DRY → Don't Repeat yourself

Q) Nth fibonacci number

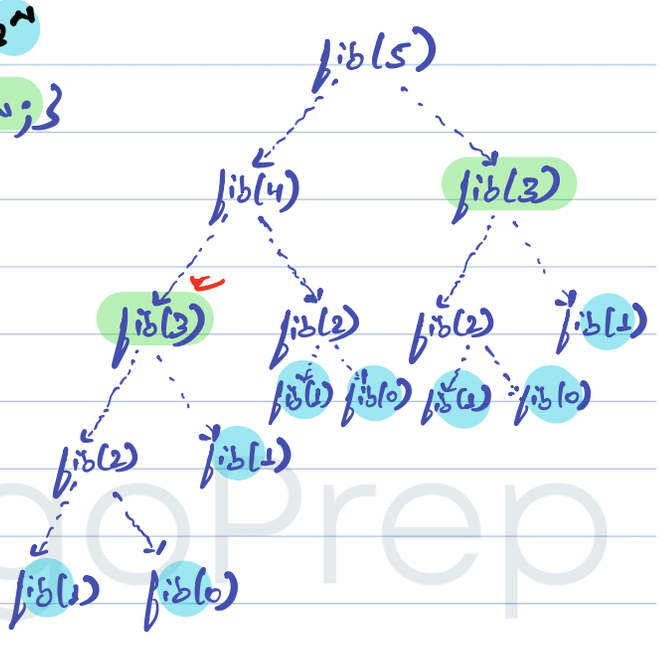
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 |

```
int fib (int N){
    if (N==0||N==1){return N;}

    int a = fib (N-1);
    int b = fib (N-2);

    return a+b;
}
```

T.C: $2^N$



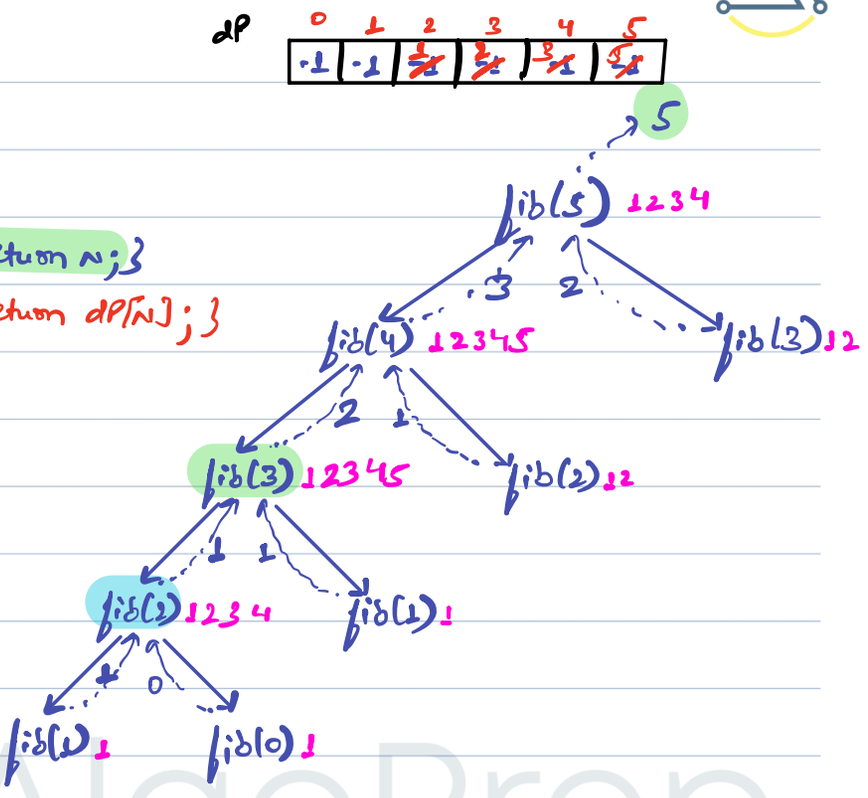
```
int dp[N+1] = {-1};

int fib (int N){
1  if (N==0||N==1){return N;}
2  if (dp[N] != -1){ return dp[N]; }

3  int a = fib (N-1);
4  int b = fib (N-2);

5  dp[N] = a+b;
6  return a+b;

}
```




T.C: O(N)

S.C: O(N)

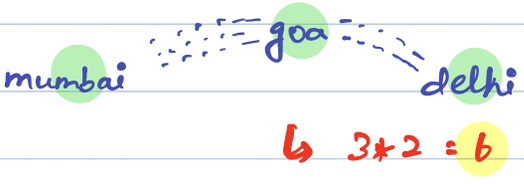
→ Optimal Substructure

↳ you can divide the problem into subproblem.

→ overlapping subproblems

↳ same problem repeating

// quick questions




↳ 3*2 = 6


⇒ 6+9 = 15

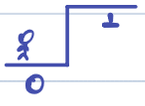
↳ 3*3 = 9

// N Stairs

↳ Given N, how many ways we can go from 0 - Nth stair.

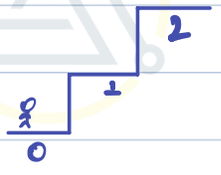
Note: you can take steps of length 1 or 2.

N = 1




ways = 1

N = 2

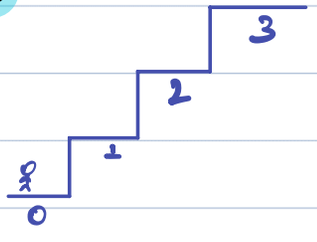



ways = 2

1 1

2

N = 3



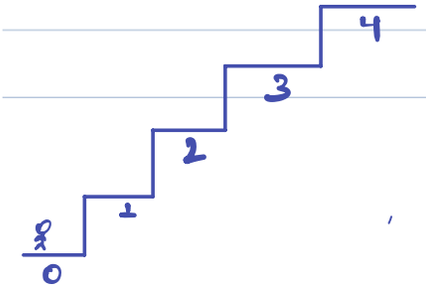

ways = 3

1 1 1

1 2

2 1

N = 4




ways = 5

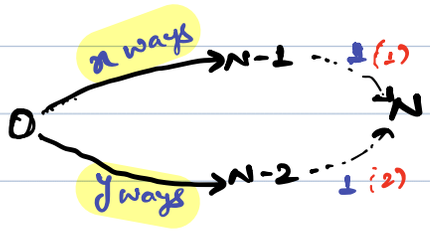
1 1 1 1

1 1 2

1 2 1

2 1 1

2 2




$x*1 + y*1 \Rightarrow x+y$

No. of ways (N) = No. of ways (N-1) + No. of ways (N-2)

N = 1 → 1

N = 2 → 2

```
int dp[N+1] = {-1};

int Stairs(int N){
1  if (N==1||N==2){return N;}
2  if (dp[N] != -1){ return dp[N]; }

3  int a = Stairs(N-1);
4  int b = Stairs(N-2);
5  dp[N] = a+b;
6  return a+b;

}
```

Break till 9:25 PM

## 11 Steps for DP

- (i) optimal substructure
- (ii) overlapping subProblem

(i) **dp state :** what are we solving → fib(i)

(ii) **recurrence relation**

↳ relation bet$^n$ Problem and subProblem.

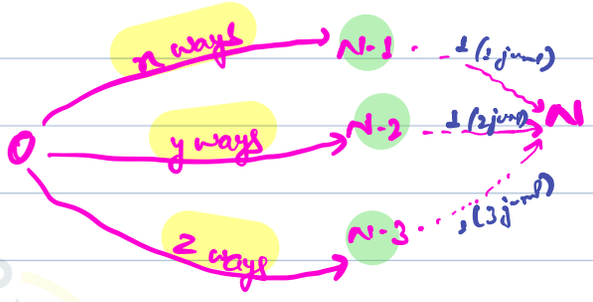
(iii) **dp table**

↳ where we store the answer.

## N Stairs

↳ Given N, how many ways we can go from 0 - Nth stairs.

Note: you can take steps of length 1 or 2 or 3






### Golden Rule of Recursion:

↳ No. of Calls = No. of Choices.

Q) Find minimum number of Perfect Squares required to Sum = N.

N=6

↳ $1^2+1^2+1^2+1^2+1^2+1^2 = 6$

$2^2+1^2+1^2 = 3$

N=10

↳ $1^2+1^2+\ldots\ldots-1^2 = 10$

↳ $2^2+2^2+1^2+1^2 = 4$

↳ $3^2+1^2 = 2$

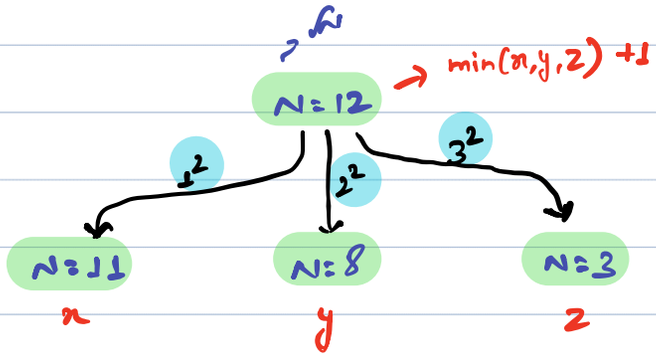
N=9

↳ $3^2 = 1$

N=12

↳ $3^2+1^2+1^2+1^2 = 4$

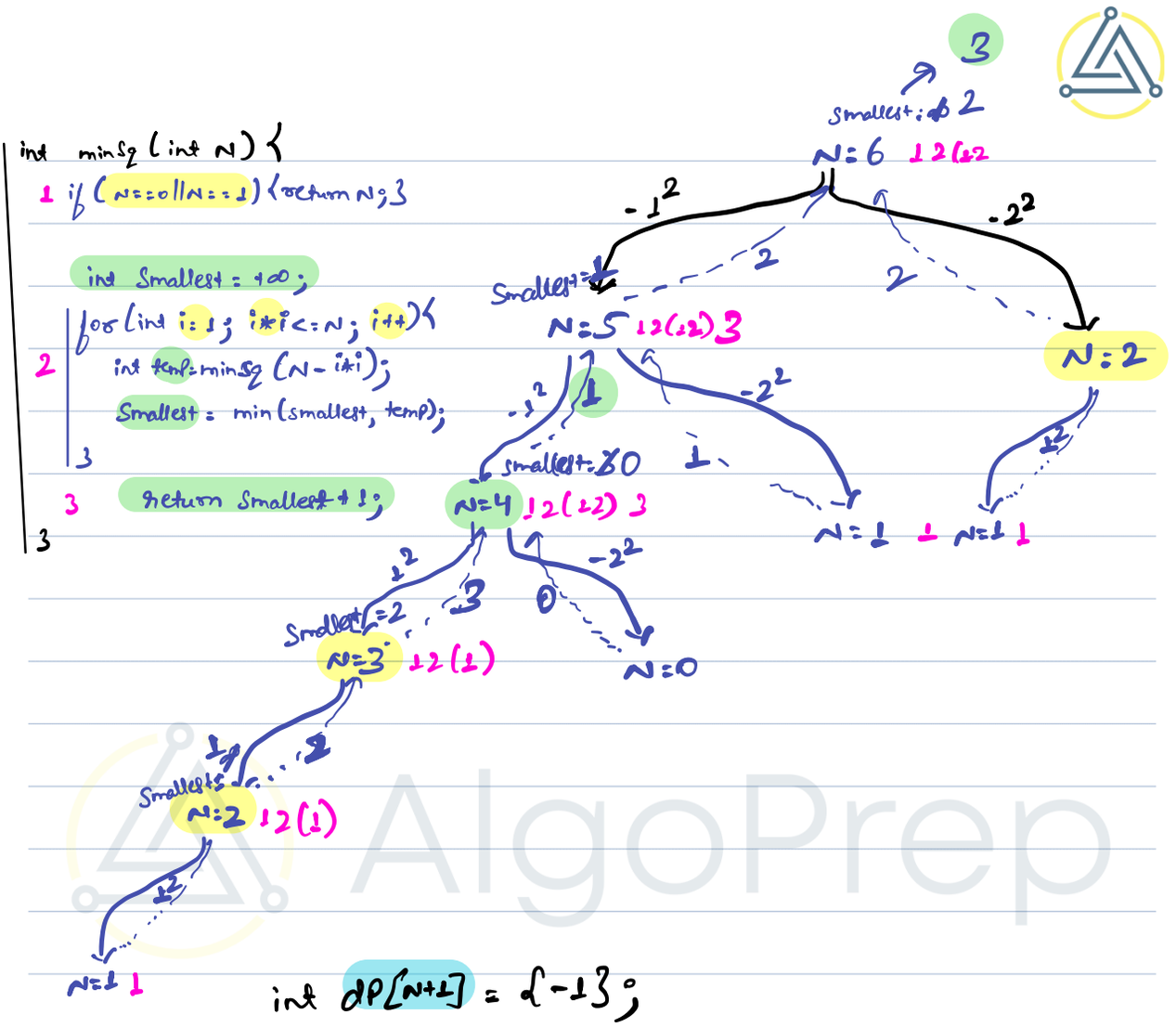
↳ $2^2+2^2+2^2 = 3$




```
int minSq (int N) {
    if (N==0||N==1) {return N;}

    int Smallest = +oo;
    for(int i=1; i*i<=N; i++){
        int temp=minSq (N-i*i);
        Smallest = min(smallest, temp);
    }

    return Smallest+1;
}
```






```
int dp[N+1] = {-1};

int minSq (int N, int[] dp) {
    if (N==0||N==1) {return N;}
    if (dp[N] != -1) {return dp[N];}

    int Smallest = +∞;
    for(int i=1; i*i<=N; i++){
        int temp=minSq (N-i*i, dp );
        Smallest = min(smallest, temp);
    }

    dp[N] = Smallest + 1;
    return Smallest+1;
}
```

T.C:

S.C:

```
int Smallest = +∞;
for(int i=1; i*i<=N; i++){
    int temp=minSq(N-i*i);
    Smallest = min(smallest, temp);
}
```

```
int Smallest = +∞;
int temp1 = minSq(N-1^2);
Smallest = Math.min(smalles, temp1);
int temp2 = minSq(N-2^2);
Smallest = Math.min(smallest, temp2)
int temp3 = minSq(N-3^2);
Smallest = Math.min(smallest, temp3);
return smallest + 1;
```