Today's agenda

- Dynamic Programming Intro
- when to use DP
- steps for DP
- # N Stairs → {1,2}
- Sqrt

AlgoPrep

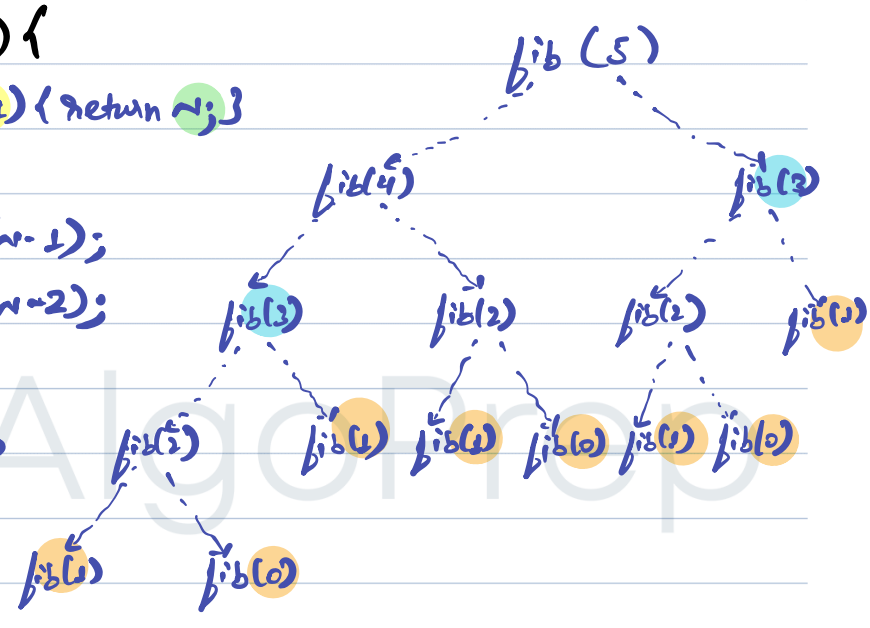
→ DRY → Don't repeat yourself.

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: $O(2^n)$



$n = 5$

int dp[n+1] = {-1}

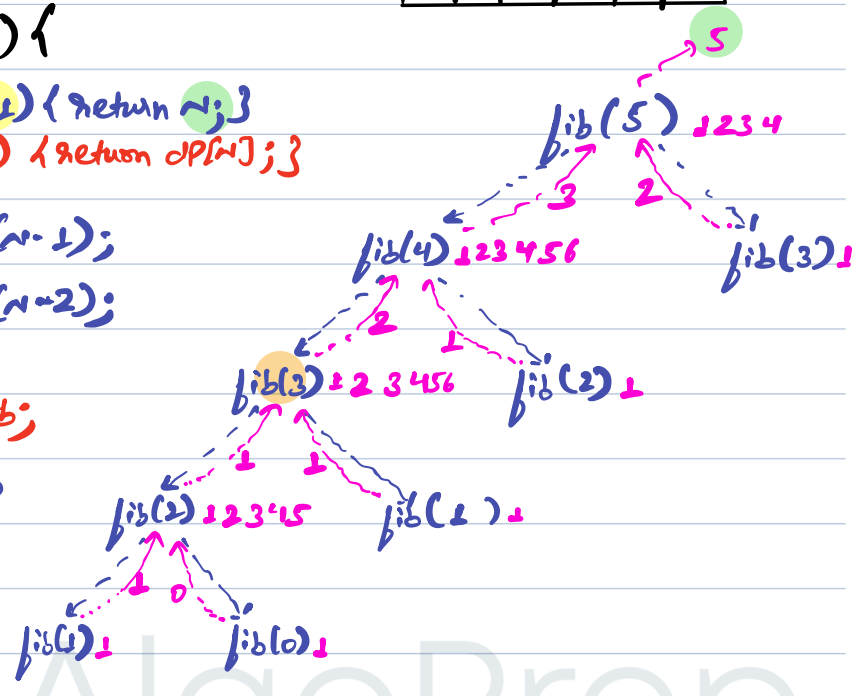
dp

0	1	2	3	4	5
-1	-1	~~-1~~ 1	~~-1~~ 2	~~-1~~ 3	~~-1~~ 5

```
int fib (int n){
    if (n==0||n==1) { return n; }
    if (dp[n] != -1) { return dp[n]; }
    int a = fib(n-1);
    int b = fib(n-2);
    dp[n] = a+b;
    return a+b;
}
```

T.C: O(n)

S.C: O(n)



→ **Optimal Substructure:**

↳ you can divide the Problem into SubProblem.

→ **overlapping SubProblem**

↳ repetition of same Problem.

AlgoPrep

// quick questions

↳ 3*2 = 6

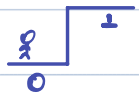
AlgoPrep

// N Stairs

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

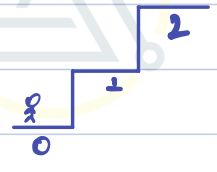
Note: you can take steps of length 1 or 2.

N = 1



way = 1

N = 2

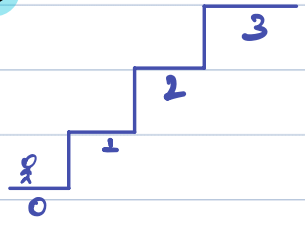


ways = 2

1 1

2

N = 3



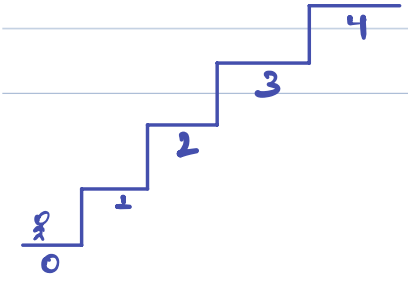
ways = 3

1 1 1

1 2

2 1

N = 4



ways = 5

1 1 1 1

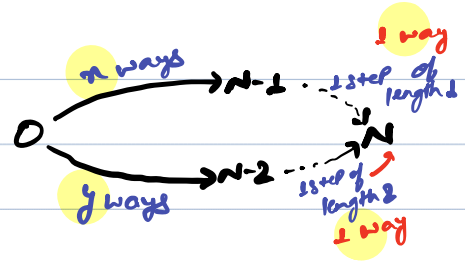
1 1 2

1 2 1

2 1 1

2 2

AlgoPrep



Total ways = $x*1 + y*1$

$= x + y$

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

$N=1 \rightarrow 1$

$N=2 \rightarrow 2$

//Psuedo code

```
int dP[N+1] = {-1}

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

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

    dP[N] = a+b;
    return a+b;
}
```

// 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

Total no. of ways = $x*1 + y*1 + z*1$

$= x+y+z$.

Golden rule of Recursion:

↳ No. of Calls = No. of Choices.

Break till 9:30 pm

11 Steps for DP

(i) optimal substructure
(ii) overlapping sub problem.

(i) **dP state:** what are we trying to solve at one instance.

(ii) **recurrence relation:**
↳ relation betn problem and subproblem.
+
base case

(iii) **dP table**
↳ where we are going to store answer of 1 instance.

AlgoPrep

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

↳ $2^2+2^2+1^2+1^2 = 4$

↳ $3^2+1 = 2$

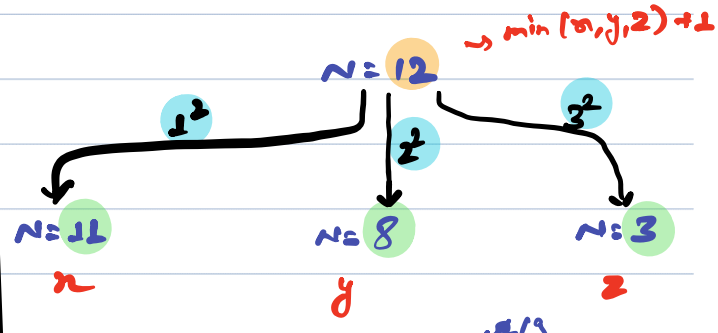
N=9

↳ $3^2 = 1$

N=12

↳ $3^2+1^2+1^2+1^2 = 4$

↳ $2^2+2^2+2^2 = 3$



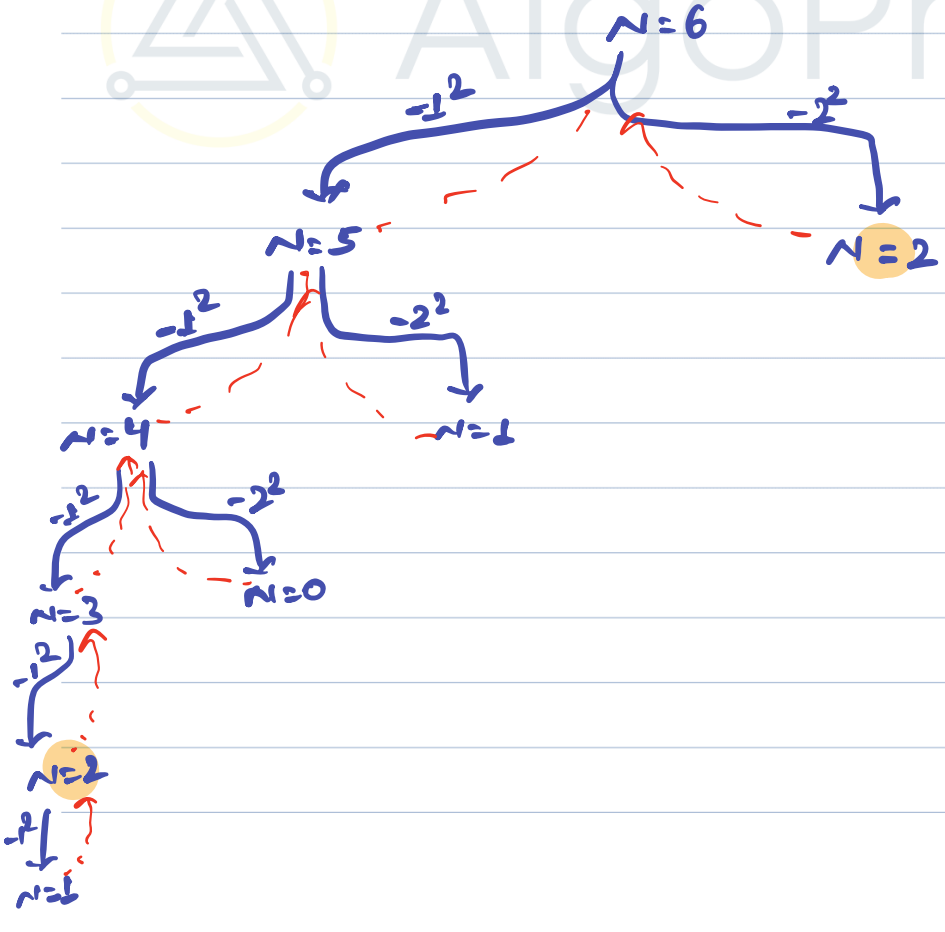

```
int minsq (int N) {
    if (N==0 || N==1){return N;}

    int smallest = +∞;
    int x = minsq (N-1²);
    smallest = mathmin(smallest, x);
    int y = minsq (N-2²);
    smallest = mathmin(smallest, y);
    int z = minsq (N-3²);
    smallest = mathmin(smallest, z);
    int w = minsq (N-4²)
    smallest = mathmin(smallest, w);

    return smallest + 1;
}
```

AlgoPrep

```
int dP[N+1] = {-1};
int minSq (int N) {
    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);
        smallest = mathmin(smallest, temp);
    }
    dP[N] = smallest + 1;
    return smallest + 1;
}
```

T.C: $O(\sqrt{N} * N)$

S.C: $O(N)$ + stack space $= O(N)$



```
int dP[N+1] = {-1};          dP1

int minSq (int N) {
    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);
        smallest = mathmin(smallest, temp);
    }
    dP[N] = smallest + 1;
    return smallest + 1;
}
```

AlgoPrep