Agenda

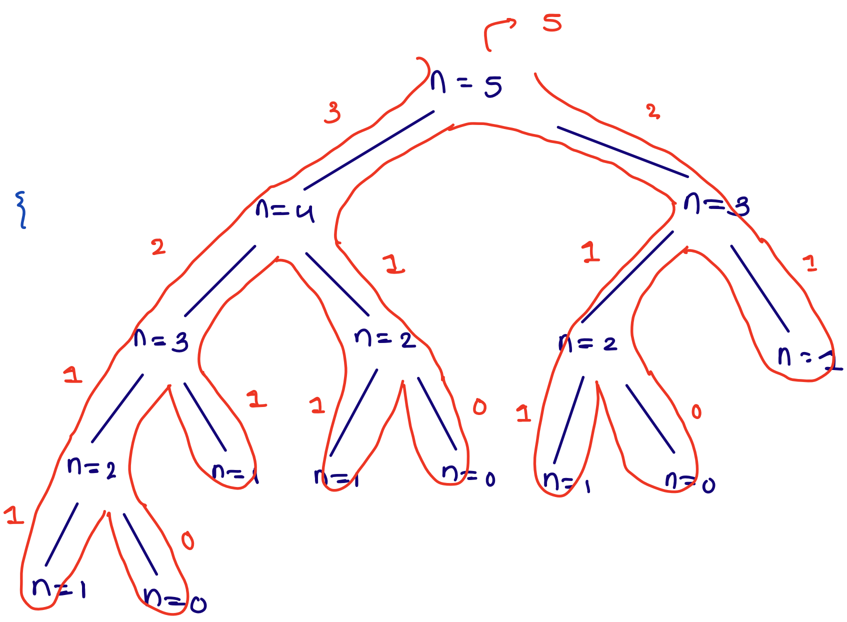
1) Introduction to DP (using fibonacci)
2) N stairs
3) min no. of squares.

fibonacci

Given n, find the n^{th} 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;
}
```



TC: $O(2^n)$

SC: $O(n)$

DP → i) solve problem with the help of sub-problem (recursion)

ii) repeatation of sub-problems

⇒ don't solve same sub-problems again & again rather once you solve a subproblem store its ans somewhere to be re-used again (DP)

Memoization: applying DP in recursive codes.

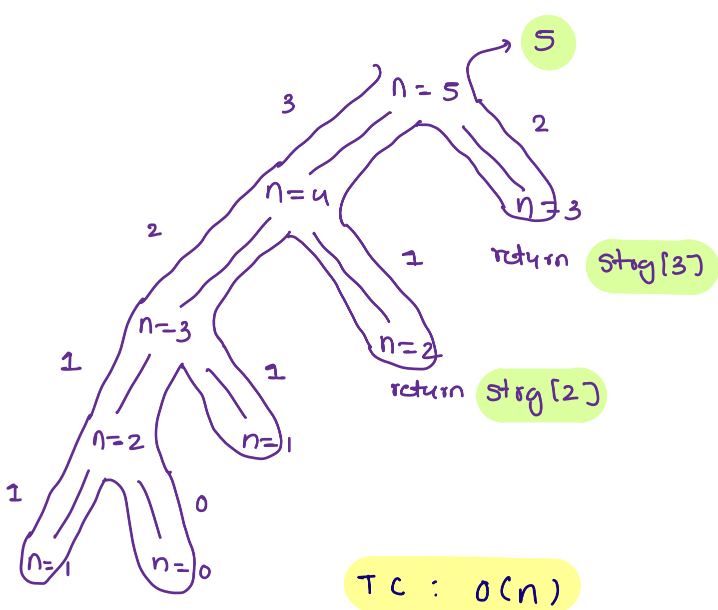
0	1	2	3	4	5	6	7	8	9	
0	1	1	2	3	5	8	13	21	34	......

n = 5

```
int[] strg = new strg[n+1];
Arrays.fill(strg, -1);
int fib (int n) {
    if (n == 0 || n == 1) {
        return n;
    }
    if( strg[n] != -1) {
        return strg[n];
    }
    int a = fib(n-1);
    int b = fib(n-2);
    strg[n] = a+b;
    return a+b;
}
```

strg

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



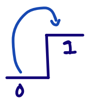
TC : O(n)

SC : O(n)

Q. Given N, find total no. of ways to go from $0^{th} \rightarrow N^{th}$ stair.

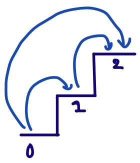
<u>Note</u>: you can take steps of length 1 & 2.

N = 1



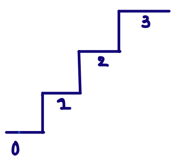
ans = 1 (1)

N = 2



ans = 2 (11, 2)

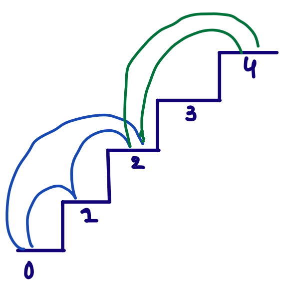
N = 3



ans = 3 (111, 12, 21)

N = 4

ans = 5

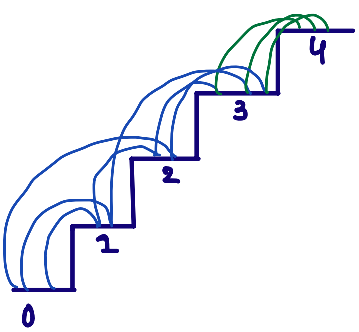


to reach 2 $\Rightarrow$ $\underbrace{11, 2}$

just add a step of 2

$\Downarrow$

112, 22 (ways to reach 4)



to reach 3 $\Rightarrow$ $\underbrace{111, 12, 21}$

just add a step of 1

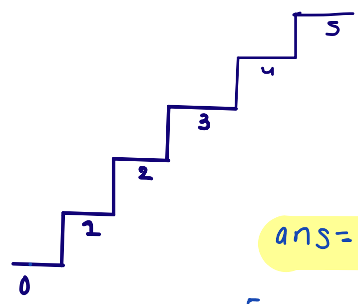
$\Downarrow$

1111, 121, 211 (ways to reach 4)

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

```
int ways (int n) {
    if(n==1 || n==2) {
        return n;
    }
    int a = ways (n-1);
    int b = ways (n-2);
    return a+b;
}
```

n = 5



ans = 8

5

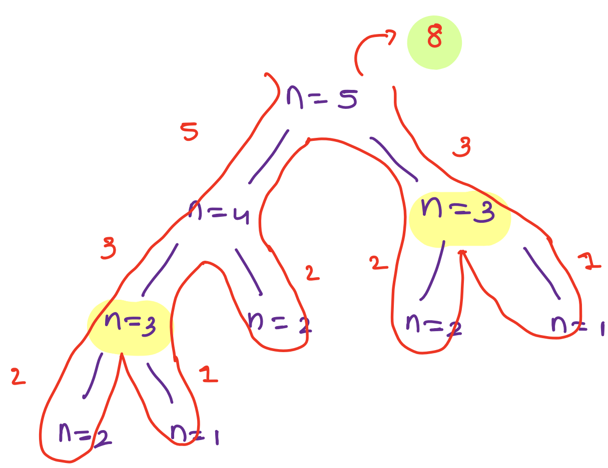
4 3

112, 22, 1111, 121, 211 — add a step of 1

111, 12, 21 — add a step of 2

1121, 221, 11111, 1211, 2111 1112, 122, 212

```
int ways (int n) {
    if(n==1 || n==2) {
        return n;
    }
    int a = ways (n-1);
    int b = ways (n-2);
    return a+b;
}
```

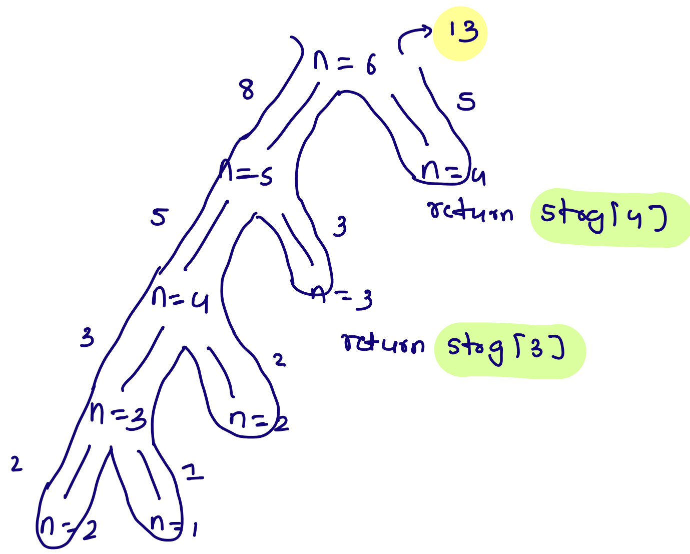


apply memoization :

```java
int[] strg = new int[n+1];
Arrays.fill(strg, -1);

int ways(int n) {
    if(n==1 || n==2) {
        return n;
    }
    if(strg[n] != -1) {
        return strg[n];
    }
    int a = ways(n-1);
    int b = ways(n-2);

    strg[n] = a+b;
    return a+b;
}
```

n = 6

-1	-1	-1	3	5	8	13
0	1	2	3	4	5	6



TC : O(n)

SC : O(n)

Steps for DP

When to apply ?

Repeated sub-problems in Recursion.

How to apply ?

Memoization

i) Create a strg of appropriate size

eg. in fib strg[x] = x^{th} fib term

in stairs strg [x] = ways to reach from 0 to x

ii) don't forget to fill strg, just before returning ans.

iii) make use of strg

Q. Find minimum count of numbers, sum of whose squares is N.

N = 6

$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$ (6)

$1^2 + 2^2 + 1^2$ (3)

ans = 3

N = 10

$1^2 + 1^2 + \dots + 1^2$ (10)

$1^2 + 2^2 + 2^2 + 1^2$ (4)

$1^2 + 3^2$ (2)

ans = 2

N = 9

......

$2^2 + 2^2 + 1^2$ (3)

3^2 (1)

ans = 1

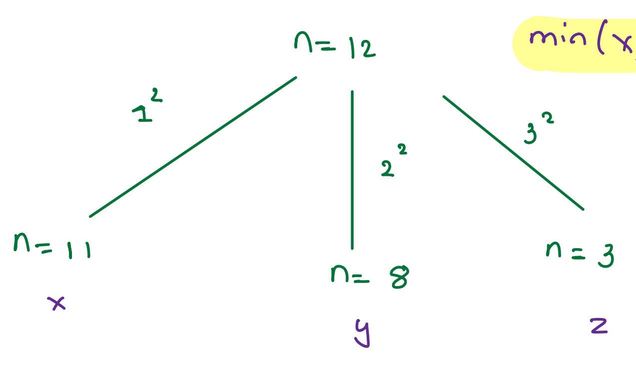
N = 12

$3^2 + 1^2 + 1^2 + 1^2$ (4)

$2^2 + 2^2 + 2^2$ (3)

ans = 3

min(x, y, z) + 1

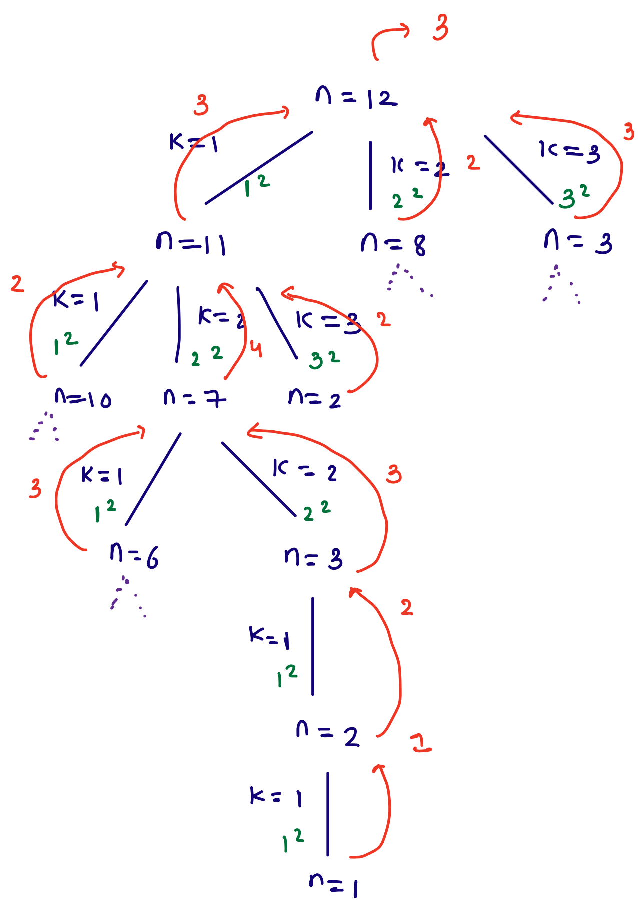


```
int  minSquare (int N) {
    if(n==0 || n==1) {
        return n;
    }
    int min = ∞;
    for(int k= 1; k*k <= n; k++) {
        int temp= minSquare(n- k*k);
        min = Math.min (min, temp);
    }
    return min+1;
}
```

Dry run

```
int  minSquare (int N) {
    if(n==0 || n==1) {
        return n;
    }
    int min = ∞;
    for(int k= 1; k*k <= n; k++) {
        int temp= minSquare(n- k*k);
        min = Math.min (min, temp);
    }
    return min+1;
}
```

$k <= \sqrt{n}$



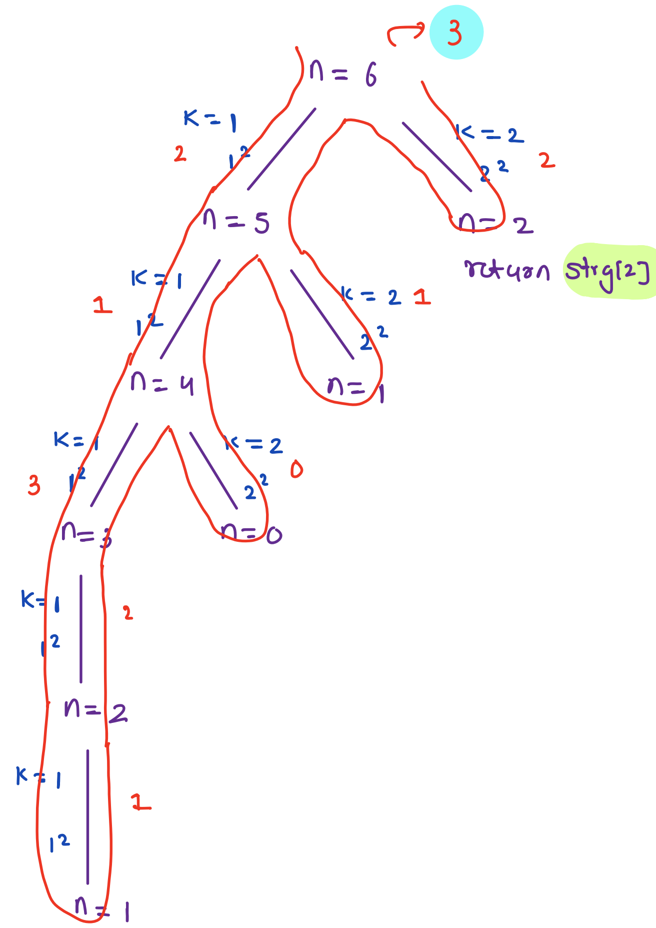
apply memoization

```
int [] strg = new int [n+1];
Arrays.fill (strg, -1);
int minSquare (int n) {
    if(n==0 || n==1) {
        return n;
    }
    if(strg [n] != -1) {
        return strg [n];
    }

    int min = ∞;              k <= √n
    for(int k= 1; k * k <= n; k++) {
        int temp= minSquare (n- k*k);
        min = Math.min (min, temp);
    }

    strg [n] = min+1;                      TC: O(n√n)
    return min+1;                          SC: O(n)
}
```

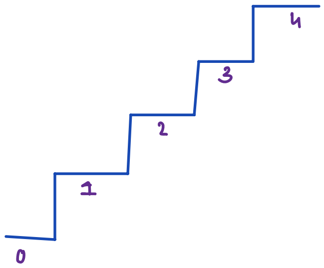
```
int[] strg = new int[n+1];
Arrays.fill(strg,-1);
int minSquare(int n) {
    if(n==0 || n==1) {
        return n;
    }
    if(strg[n] != -1) {
        return strg[n];
    }

    int min = ∞;
                                  k <= √n
    for(int k=1; k*k <= n; k++) {
        int temp = minSquare(n-k*k);
        min = Math.min(min,temp);
    }

    strg[n] = min+1;
    return min+1;

}
```

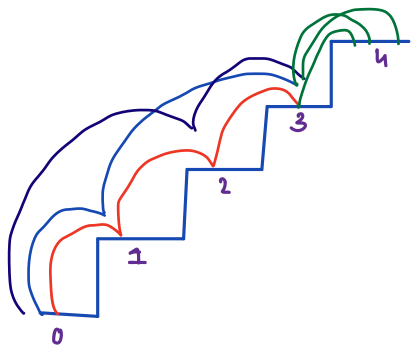
n = 6

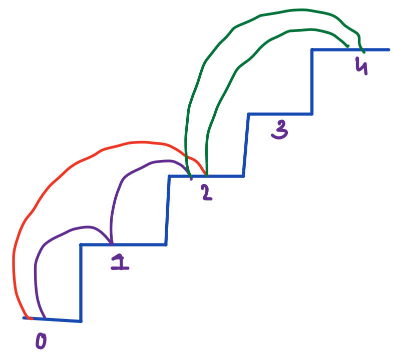
-1	-1	2	3	1	2	3
0	1	2	3	4	5	6



Doubts







ways to reach 3 => 111, 12, 21

append 1

ways to reach 4 => 1111, 121, 211

ways to reach 2 => 11, 2

append 2

ways to reach 4 => 112, 22

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