Dynamic Programming

– "Those who can't remember their past are condemned to repeat it"

Dynamic Programming – Recursion + Memoization + Tabulation

① Print n^{th} fibonacci No

0	1	1	2	3	5	8	13	21
0	1	2	3	4	5	6	7	8

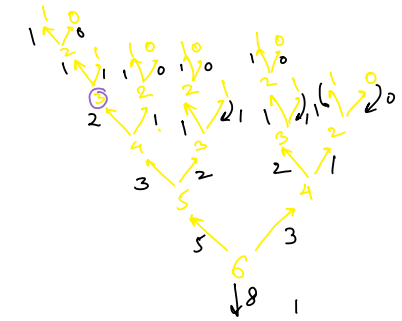
Exp: Print n^{th} fib no fib(n)

faith: → fib(n-1)
→ fib(n-2)

Meeting Expectation with faith: ans = fib(n-1) + fib(n-2)
return ans;

$T(n) = T(n-1) + T(n-2) + k$

Recursion

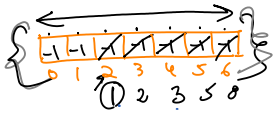


```
if(n == 0 || n == 1){
    return n;
}

int fibn1 = fibRec(n - 1);
int fibn2 = fibRec(n - 2);

int finb = fibn1 + fibn2;
return finb;
```

$T(n) = 2T(n-1) + k$

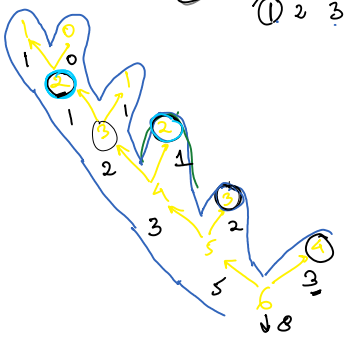
$O(2^n)$ Exponential

DP Table



O(n)

```
public static int fibMem(int n, int[] dp){
    if(n == 0 || n == 1){
        return n;
    }

    if(dp[n] != -1){
        return dp[n];
    }

    int fibn1 = fibMem(n - 1, dp);
    int fibn2 = fibMem(n - 2, dp);

    dp[n] = fibn1 + fibn2;
    return dp[n];
}
```



DP → Multiple calls to same state
→ Overlapping subproblems
→ optimal substructure
bigger problem is dependent on smaller problem

$fib(n) = fib(n-1) + fib(n-2)$

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

dp[3] = 3^{rd} fib no

dp[n] = n^{th} fib no

① Storage & Meaning

① Storage & Meaning
② Direction
③ Travel & Solve

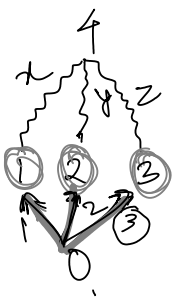
dp[n] = nth fib no

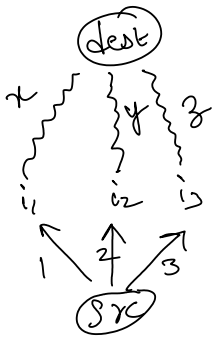
$$dp[n] = dp[n-1] + dp[n-2]$$

```
int[] dp = new int[n + 1];
dp[0] = 0; dp[1] = 1;

for(int i=2; i<=n; i++){
    dp[i] = dp[i - 1] + dp[i - 2];
}

return dp[n];
```

TC = O(n)
SC = O(n)

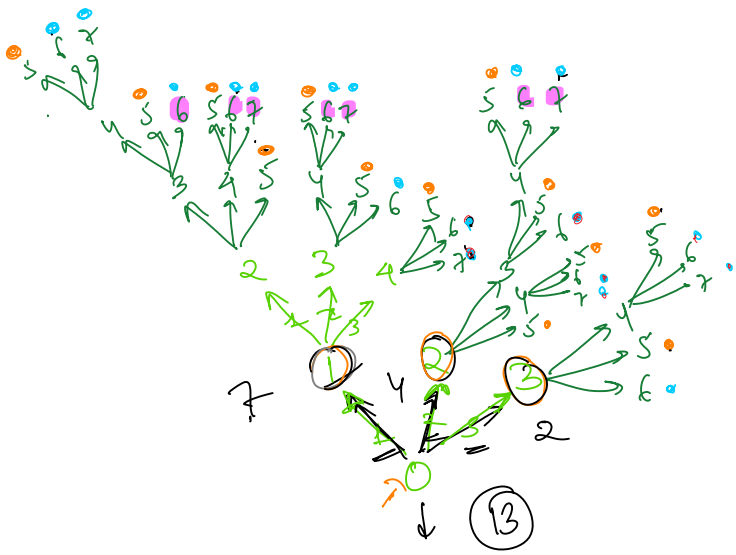
Climb Stairs





x+y+3

Exp

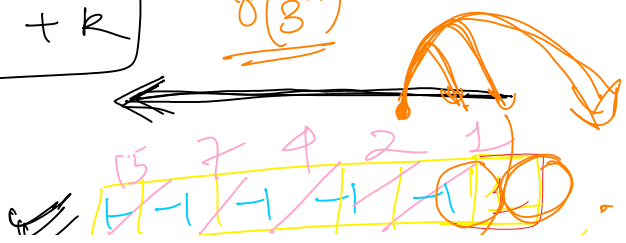


n = 5

$$T(n) = T(n-1) + T(n-2) + T(n-3) + k$$

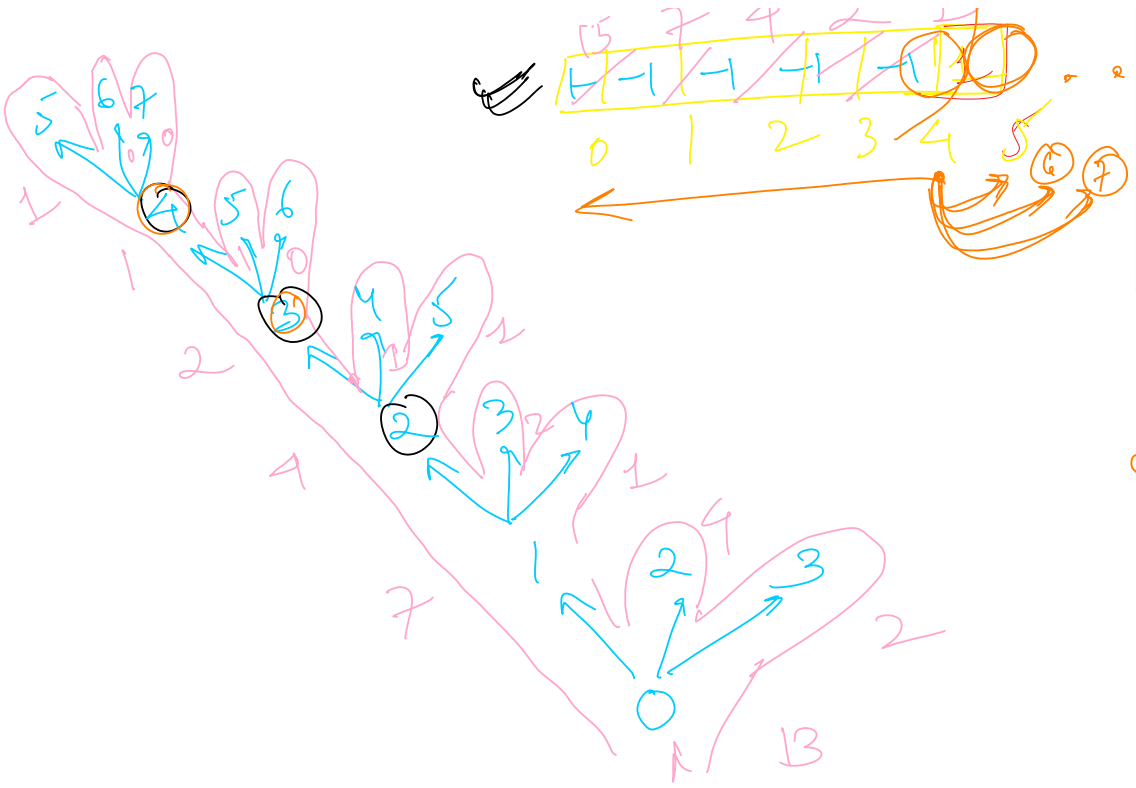
$$T(n) = 3T(n-1) + k$$

$O(3^n)$



```
public static in
    if(src > des
    if(src == de

    if(dp[src] !
        return d
```

```
t csMem(int src, int dest, int[] dp){
t) return 0;
st) return 1;

= -1){
p[src];
```



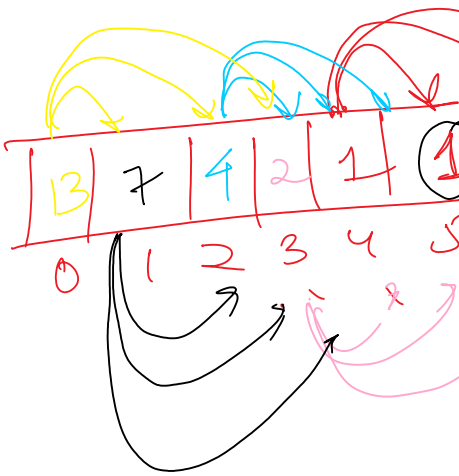
```
if(src == de

if(dp[src] !
    return d
}

int x = csMe
int y = csMe
int z = csMe

dp[src] = x
return dp[sr
}
```

dp[i] =

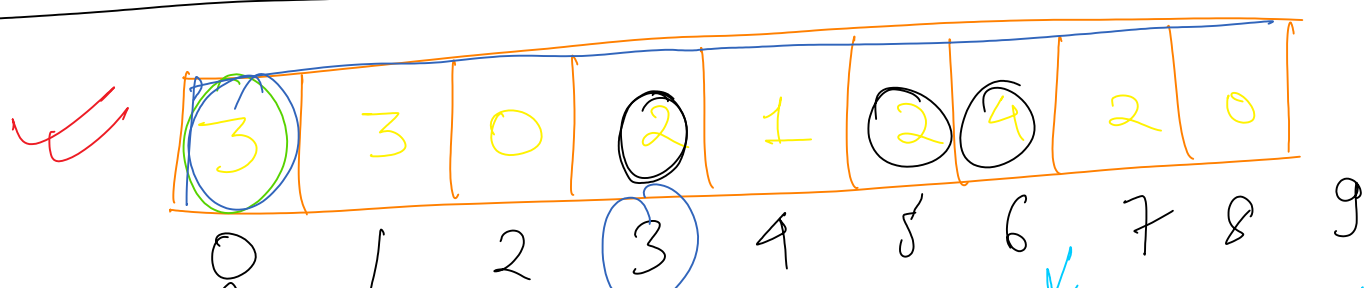
Time

Spa

```
int[] dp = new int[dest + 1];
dp[dest] = 1;
for(int i = dest - 1;  i >= 0; i--){

    dp[i] = dp[i + 1];
    if(i + 2 < dp.length){
        dp[i] += dp[i + 2];
    }

    if(i + 3 < dp.length){
        dp[i] += dp[i + 3];
    }
}

return dp[0];
```



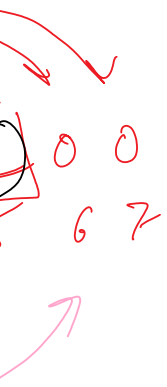
Climb stairs with variable moves

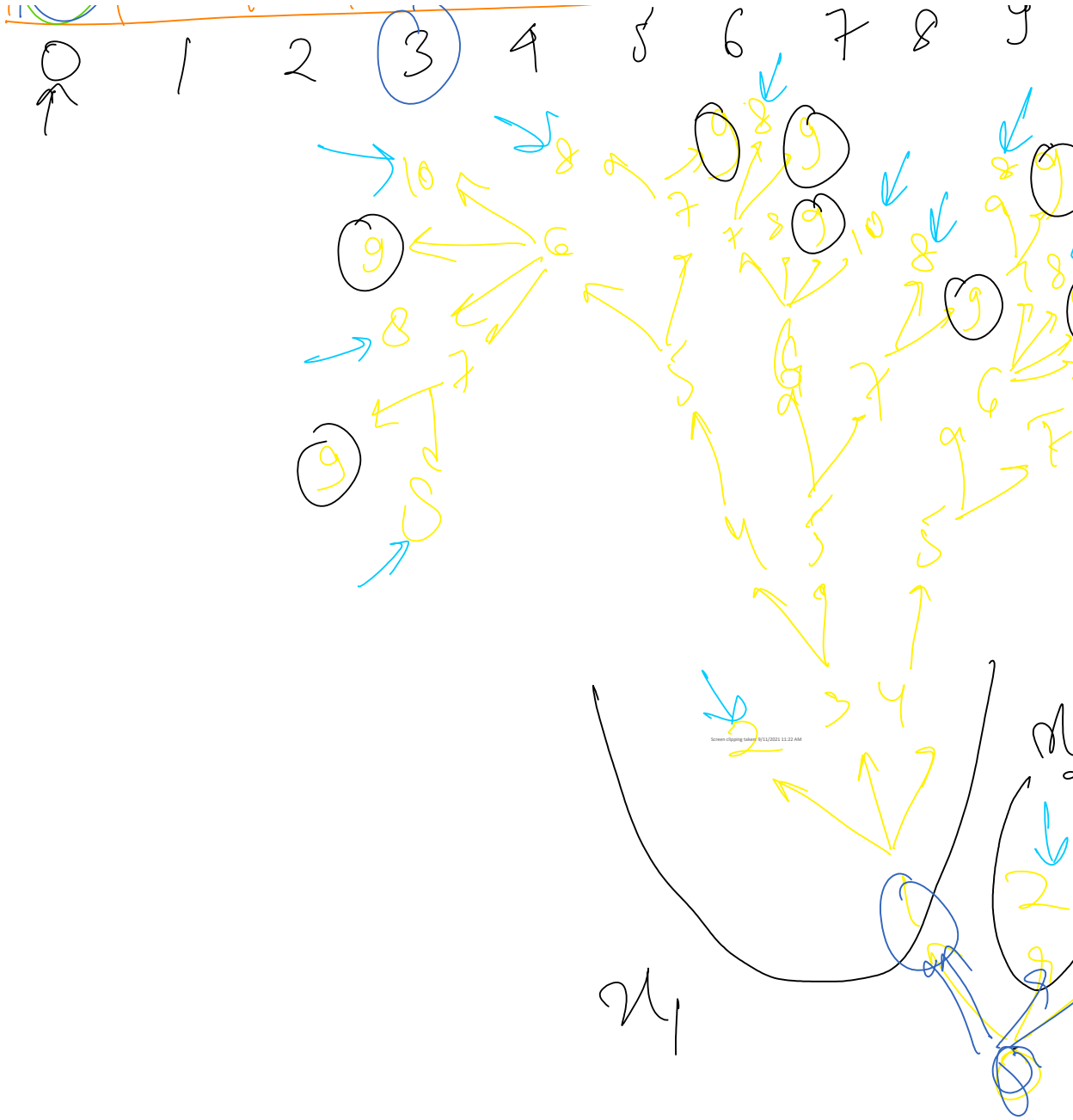


```
t) return 0;
st) return 1;

= -1){
p[src];


m(src + 1, dest, dp);
m(src + 2, dest, dp);
m(src + 3, dest, dp);

+ y + z;
c];
```

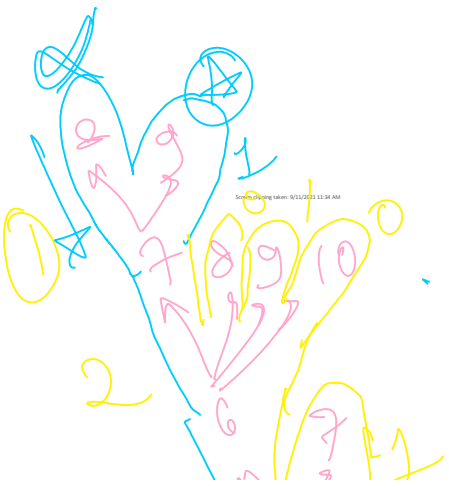
dp[it1] + dp[it2]
+ dp[it3]

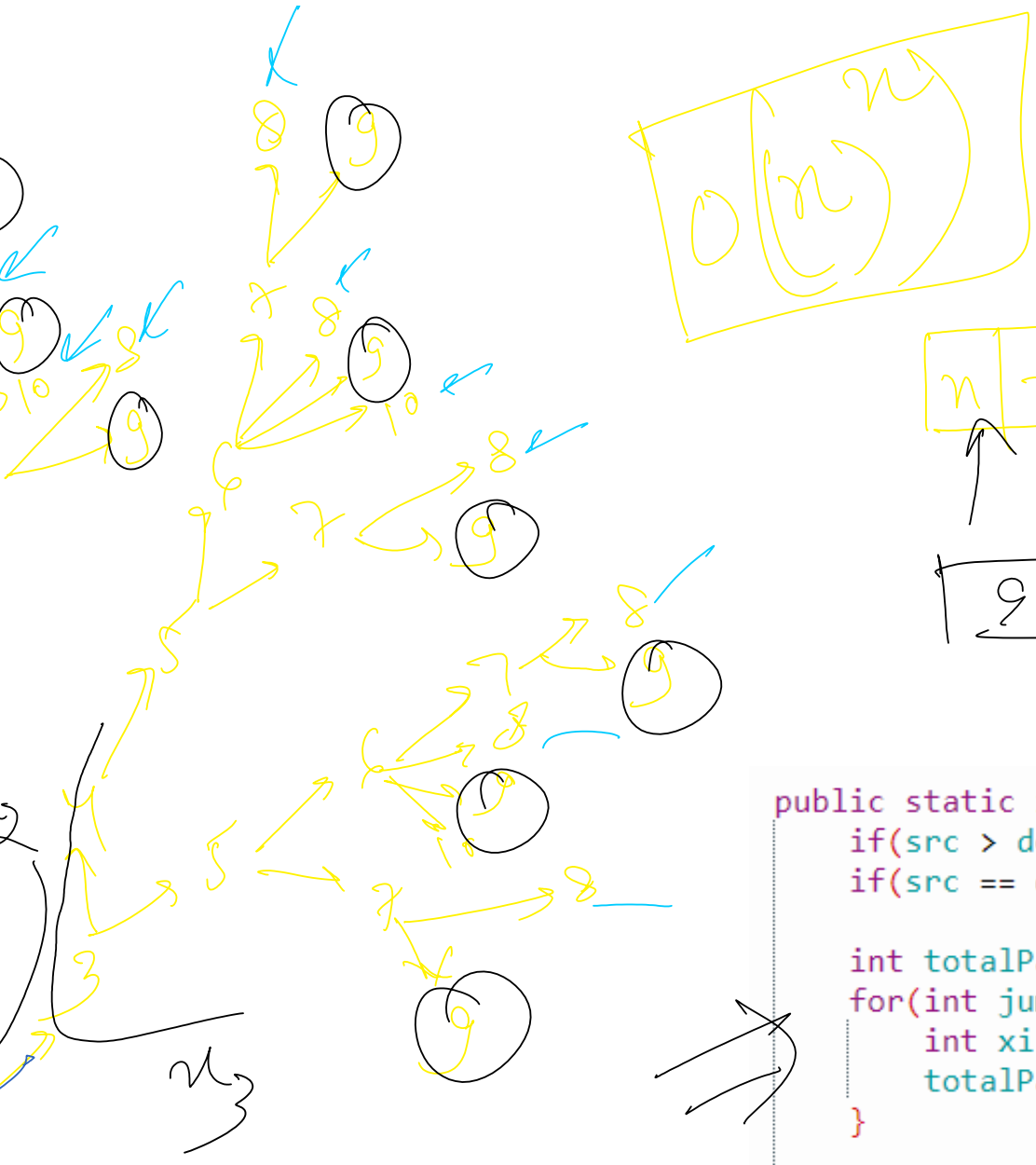
e Comp — O(n)
lce Comp — O(n)

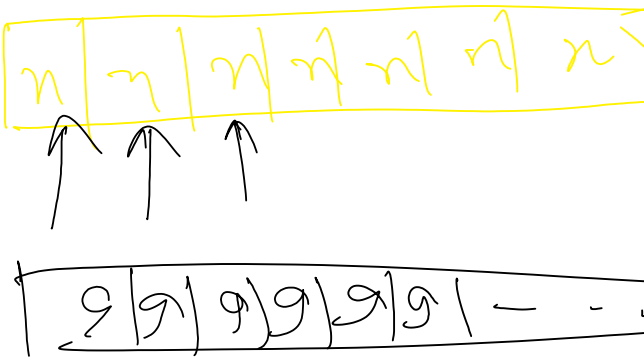




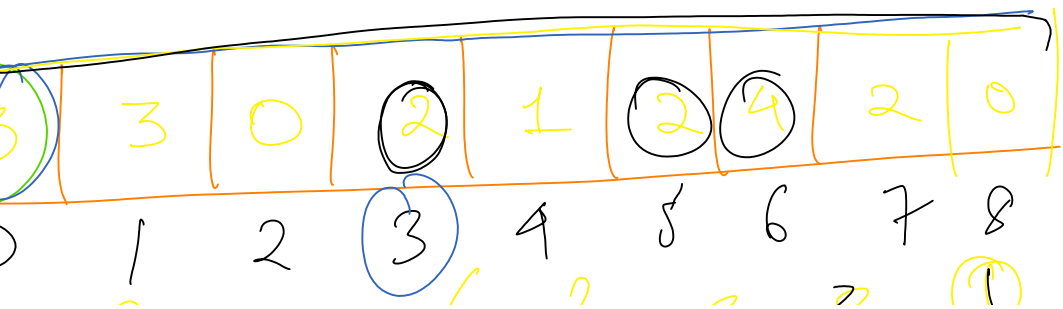
Memoization







```
public static int csvm(int src, int dest
    if(src > dest) return 0;
    if(src == dest) return 1;

    int totalPaths = 0;
    for(int jumps = 1; jumps <= arr[src]
        int xi = csvm(src + jumps, dest,
        totalPaths += xi;
    }

    return totalPaths;
}
```

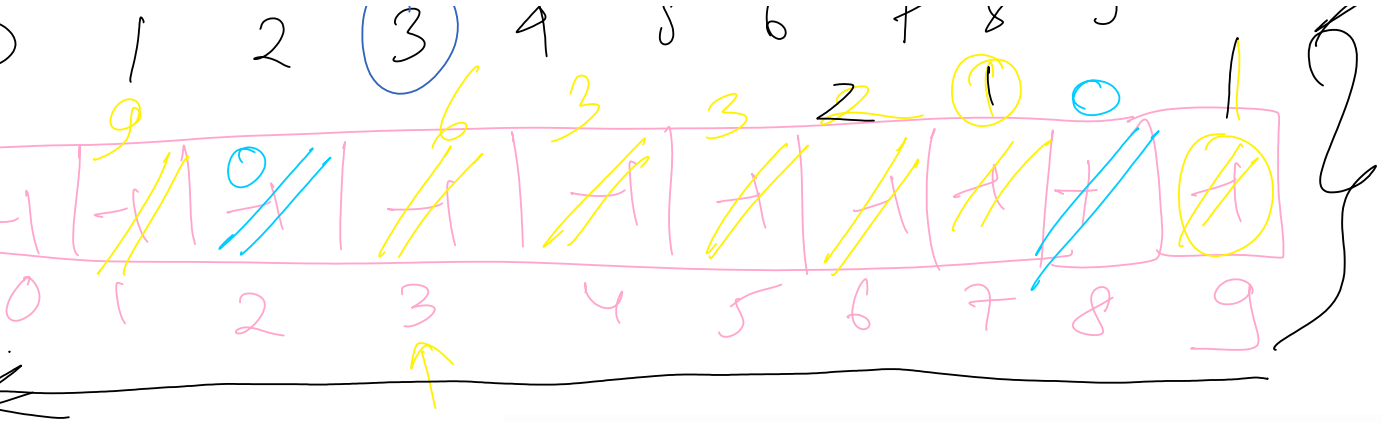


```
, int[] arr){


; jumps++){
 arr);
```

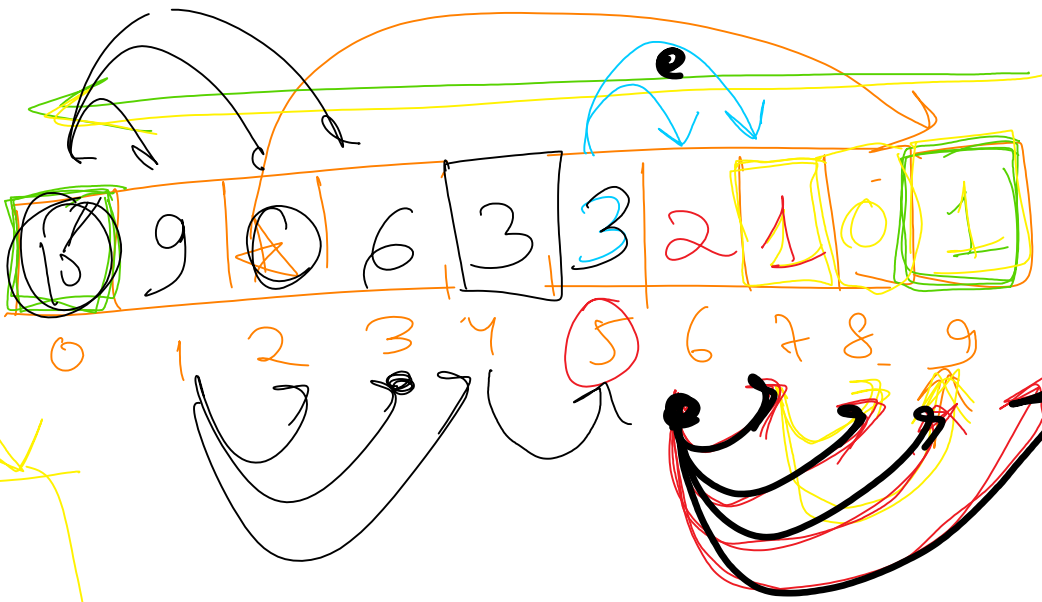
2+1=3

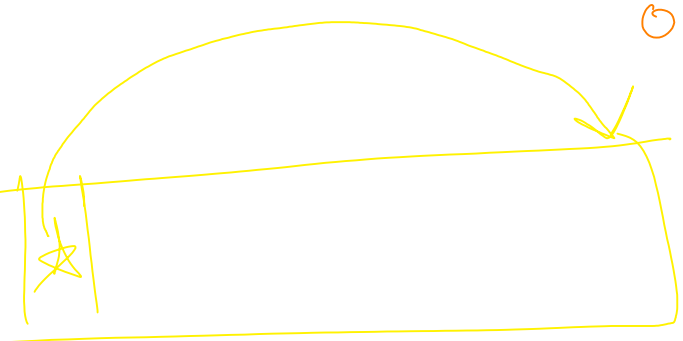
Tabular

1. Storage & Meaning
2. Direction

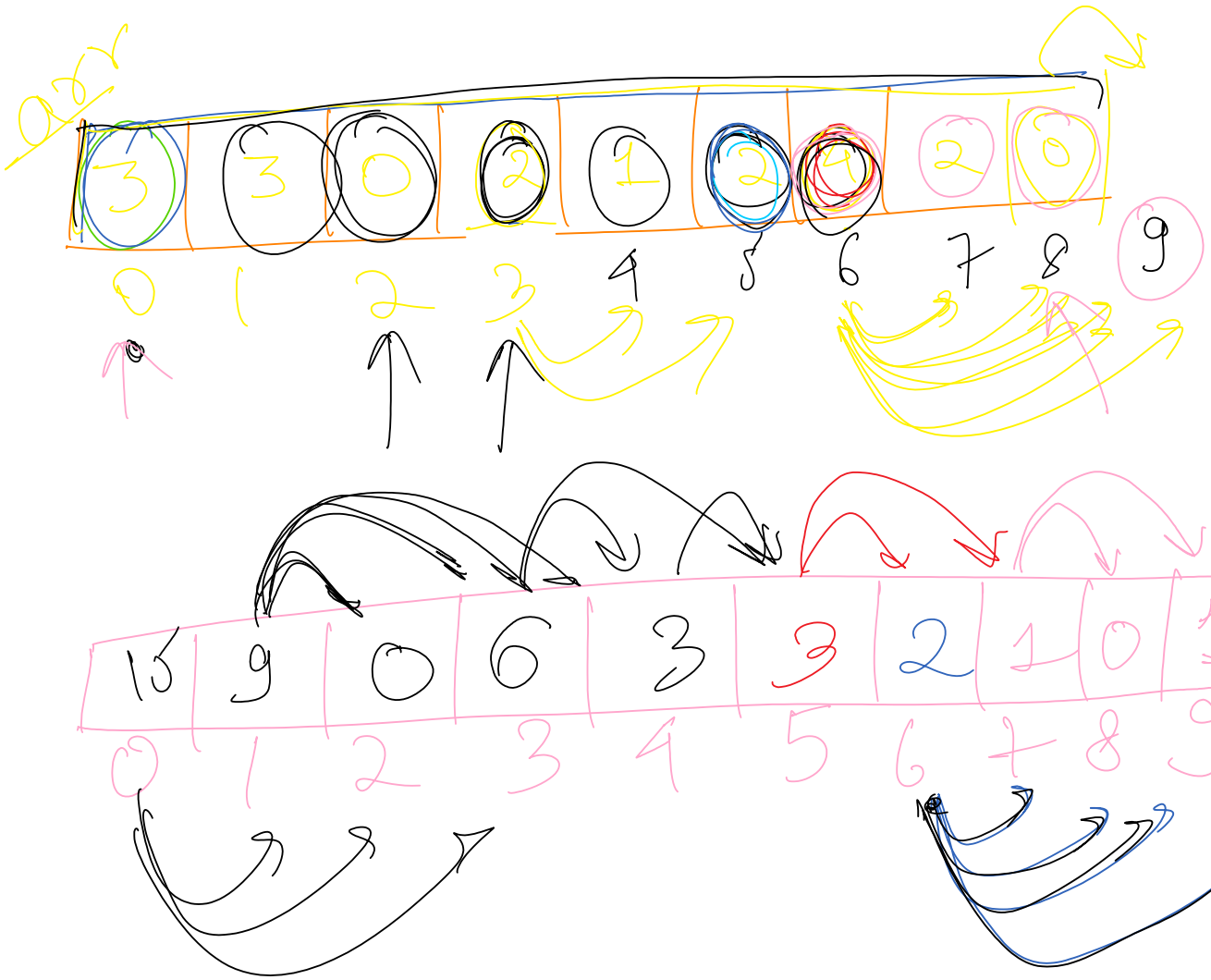


```
if(src > dest) return 0;
if(src == dest) return 1;

if(dp[src] != -1){
    return dp[src];
}

int totalPaths = 0;
for(int jumps = 1; jumps <= arr[src]; jumps++){
    int xi = csvmMem(src + jumps, dest, arr, dp);
    totalPaths += xi;
}

dp[src] = totalPaths;
return dp[src];
```





Screen clipping taken: 9/11/2021 11:58 AM

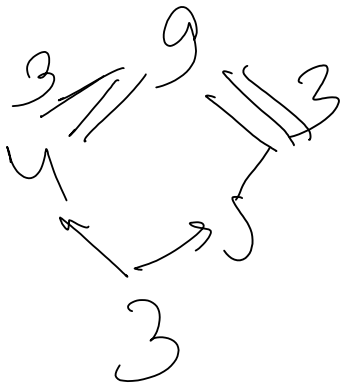


```
public s
    int
    int[
    dp[
    for(

    }

    retu
}
```

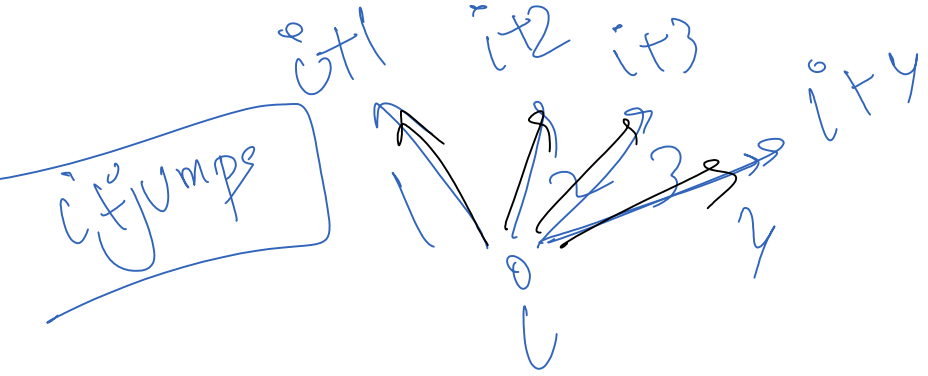
dp[i] = count of all paths from i to dest

```
static int csvmTab(int dest, int[] arr){
 n = arr.length;
[] dp = new int[n + 1];
dest] = 1;
(int i=n-1; i>=0; i--){

 int totalPaths = 0;
for(int jumps = 1; jumps <= arr[i]; jumps++){
    if(i + jumps < dp.length){
        totalPaths += dp[i + jumps];
    }
}

 dp[i] = totalPaths;


urn dp[0];
```

O(n)



9



n = O(n²)

$O(n^n)$ → $O(n^2$

exp

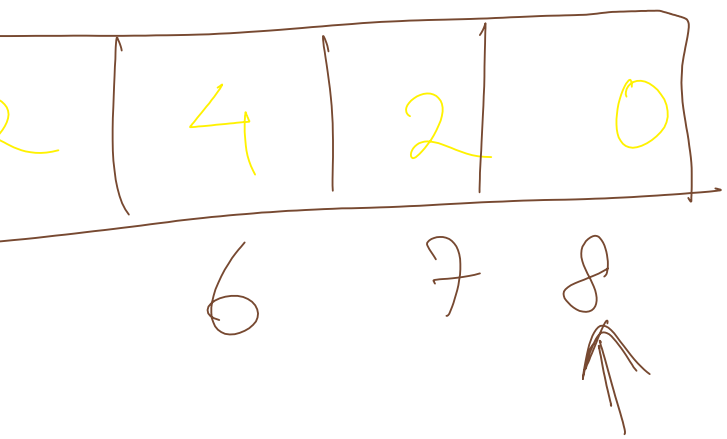
quad

Recursion

Min moves

3	3	0	2	1	
0	1	2	3	4	5

$\min(x, y, z) + 1$


4
2
0
6
7
8