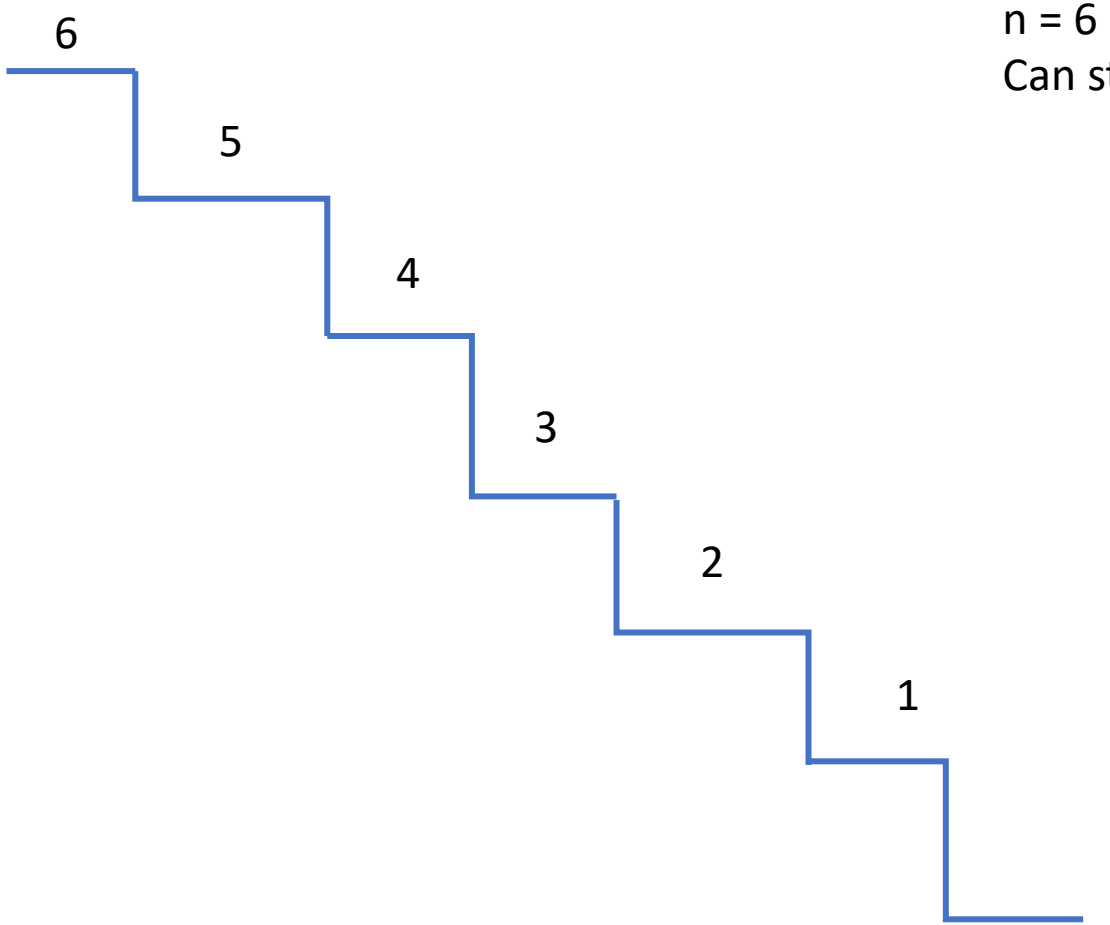
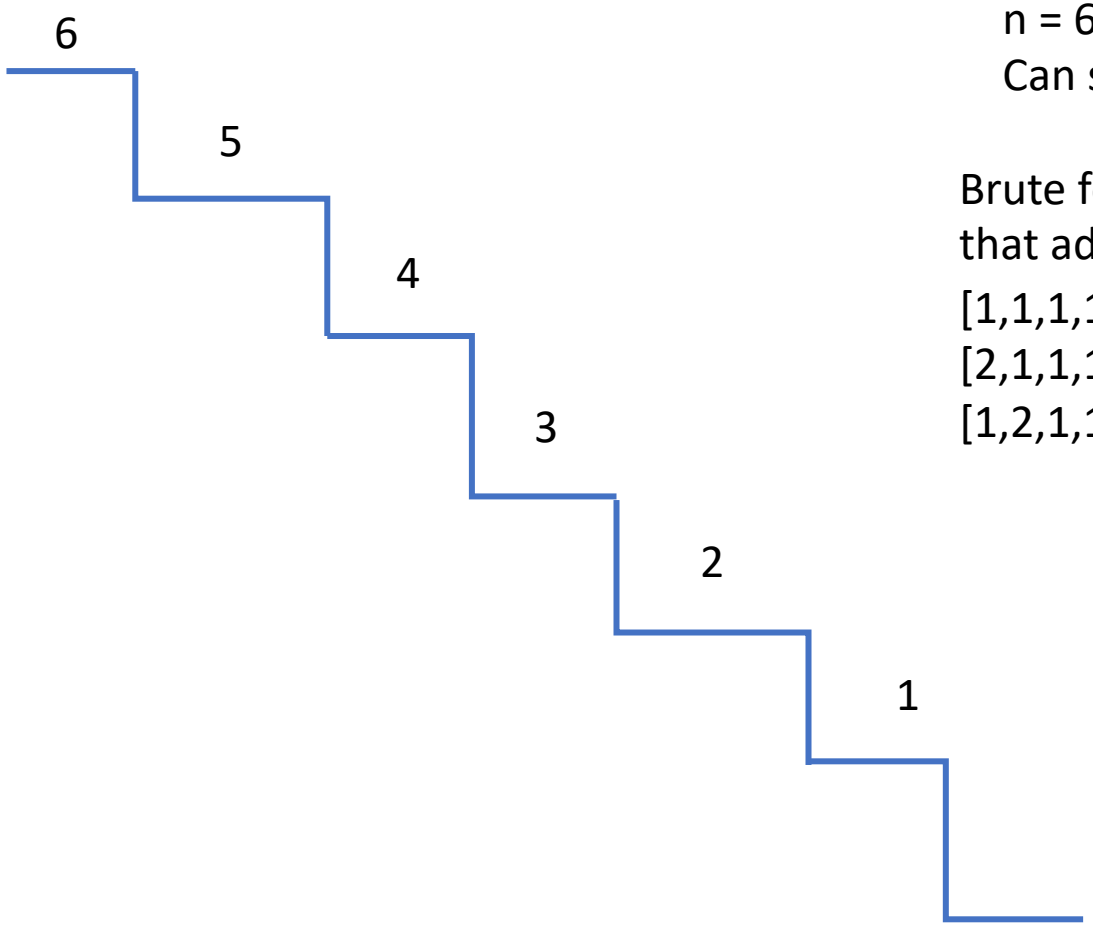
# leetcode 70

## Climbing Stairs



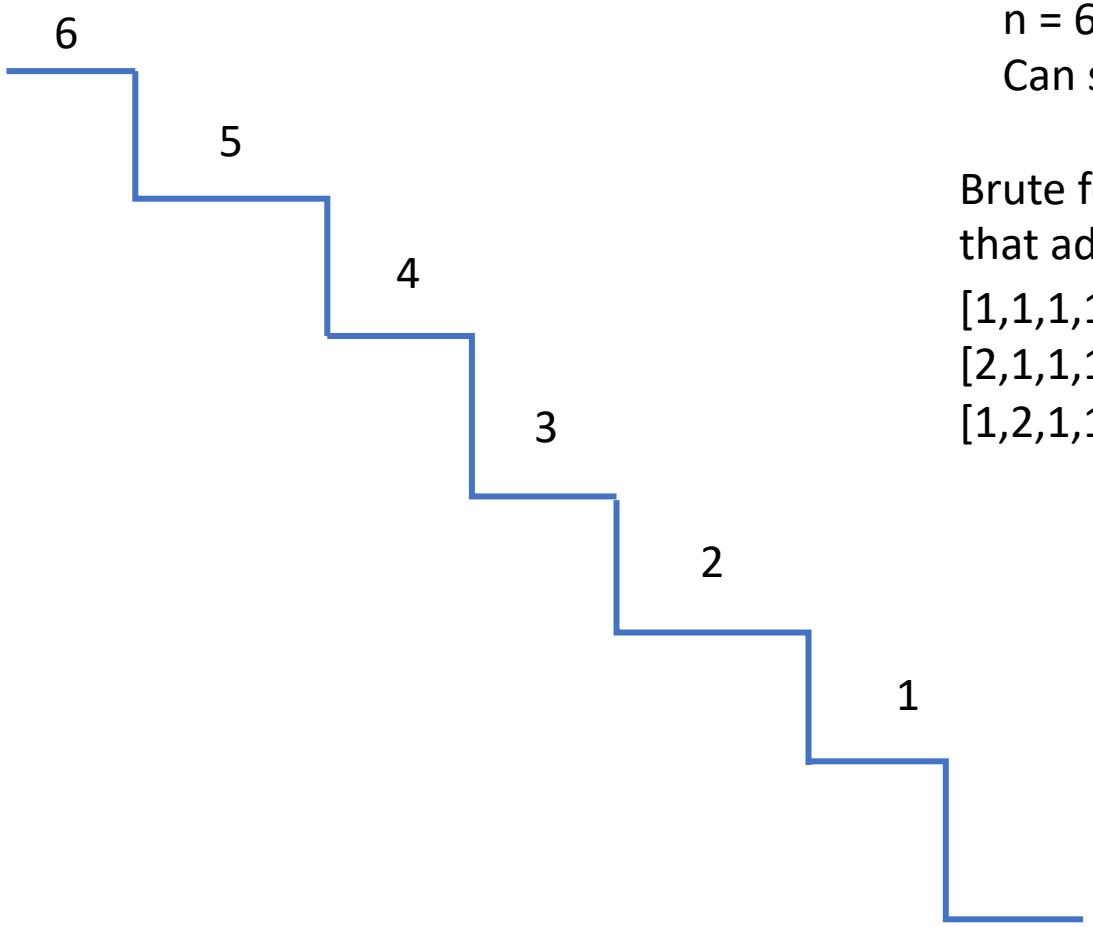

n = 6
Can step up 1 or 2 steps until top




n = 6
Can step up 1 or 2 steps until top

Brute force: list all possible combinations of 1 and 2 steps that add up to n

[1,1,1,1,1,1]
[2,1,1,1,1]
[1,2,1,1,1]

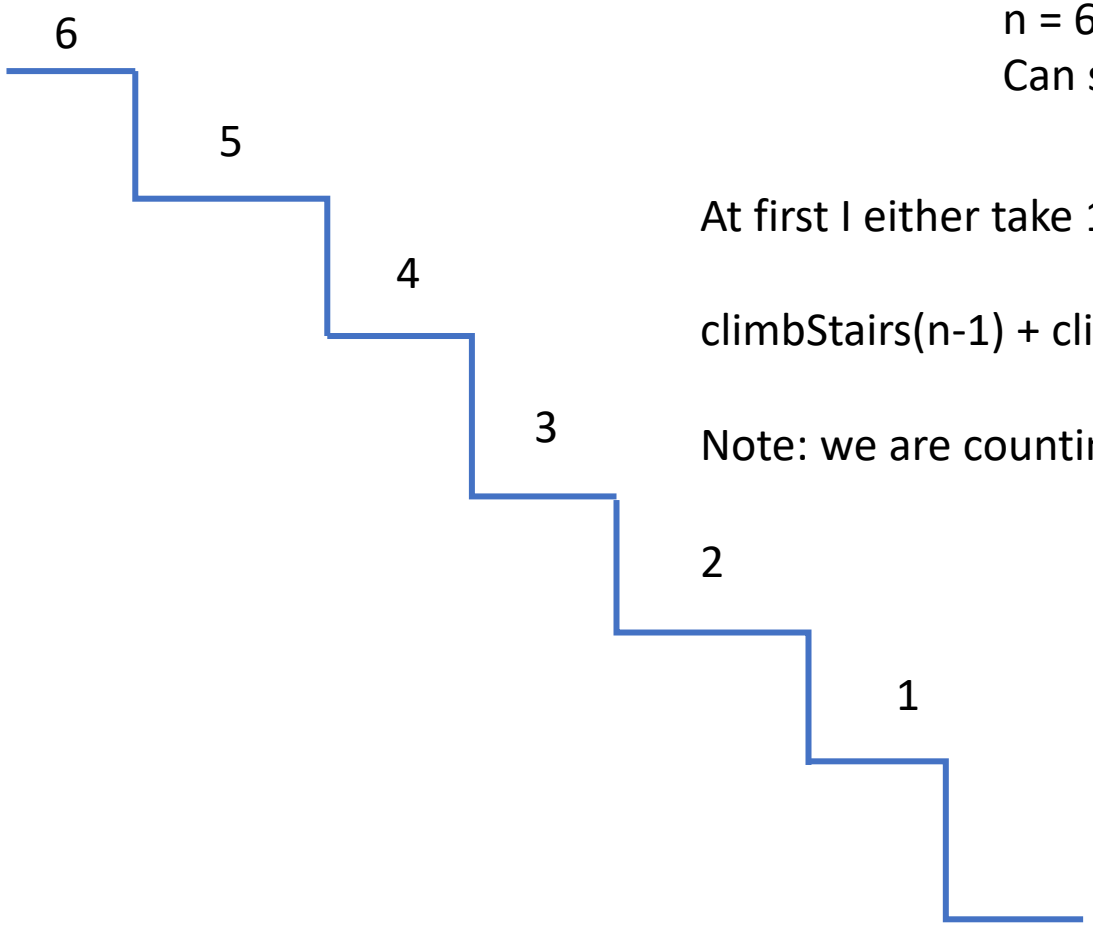



n = 6
Can step up 1 or 2 steps until top

Brute force: list all possible combinations of 1 and 2 steps that add up to n

[1,1,1,1,1,1]
[2,1,1,1,1]
[1,2,1,1,1]

Can we define an optimal solution  from optimal solution to subproblems?



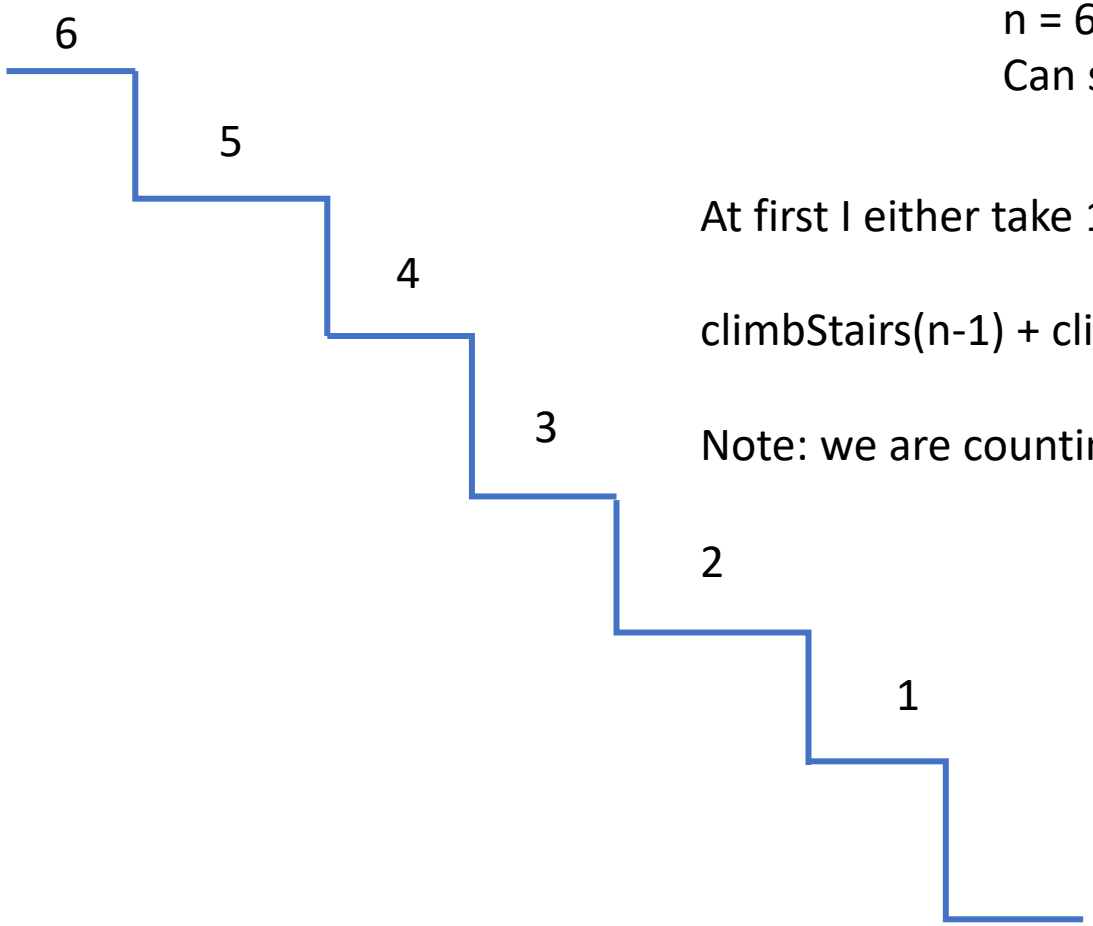

n = 6
Can step up 1 or 2 steps until top

At first I either take 1 or 2 steps, then solve the remaining problem:

climbStairs(n-1) + climbStairs(n-2)

Note: we are counting how many ways, not how many steps!

Can we define an optimal solution  from optimal solution to subproblems?




n = 6
Can step up 1 or 2 steps until top

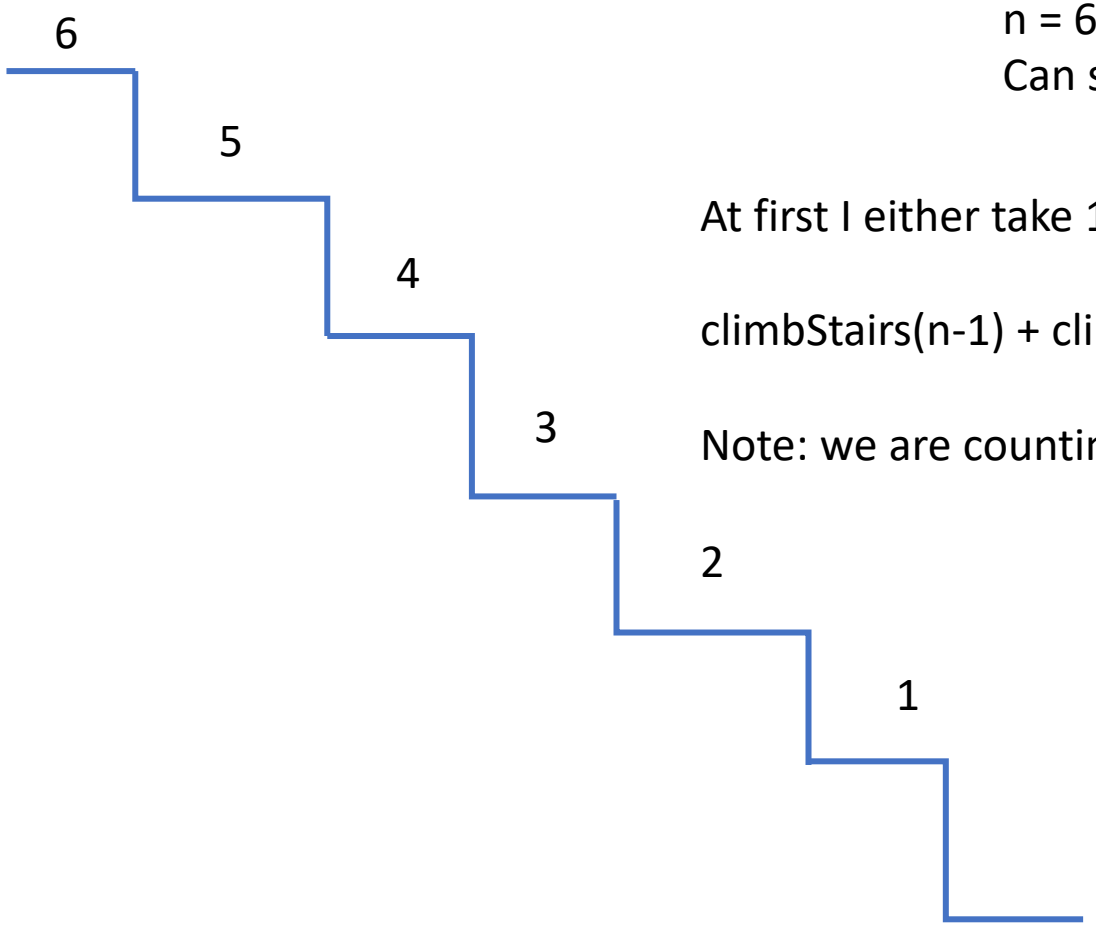
At first I either take 1 or 2 steps, then solve the remaining problem:

climbStairs(n-1) + climbStairs(n-2)

Note: we are counting how many ways, not how many steps!

```
if(n <= 2) return n
else return climbStairs(n-1) + climbStairs(n-2)
```

Can we define an optimal solution  from optimal solution to subproblems?




n = 6
Can step up 1 or 2 steps until top

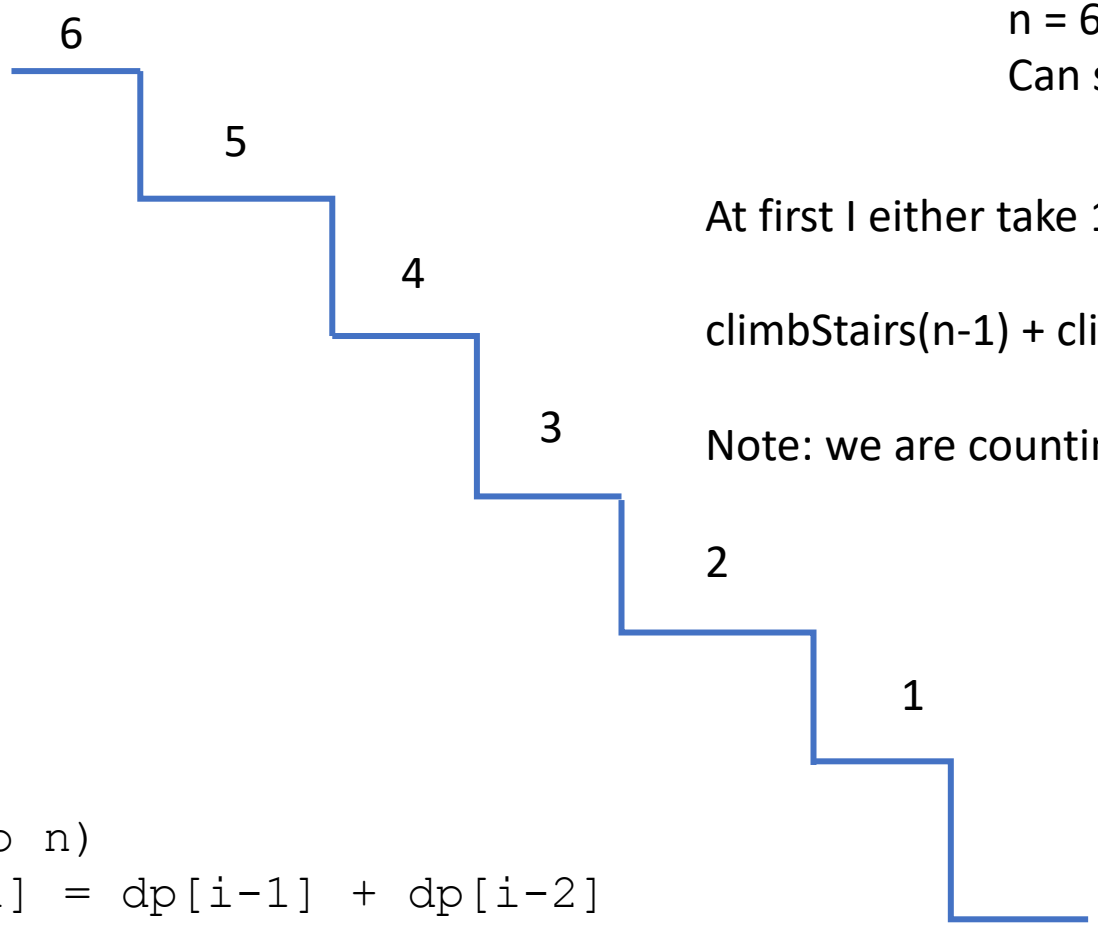
At first I either take 1 or 2 steps, then solve the remaining problem:

climbStairs(n-1) + climbStairs(n-2)

Note: we are counting how many ways, not how many steps!

```
if(n <= 2) return n
else return climbStairs(n-1) + climbStairs(n-2)
```

Can we tabulate it?




n = 6
Can step up 1 or 2 steps until top

At first I either take 1 or 2 steps, then solve the remaining problem:

climbStairs(n-1) + climbStairs(n-2)

Note: we are counting how many ways, not how many steps!

```
dp[0] = 0
dp[1] = 1
dp[2] = 2
for(i: 3 to n)
       dp[i] = dp[i-1] + dp[i-2]

return dp[n]
```

dp[i] = number of ways to reach step i

# leetcode 53

## Maximum Subarray

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

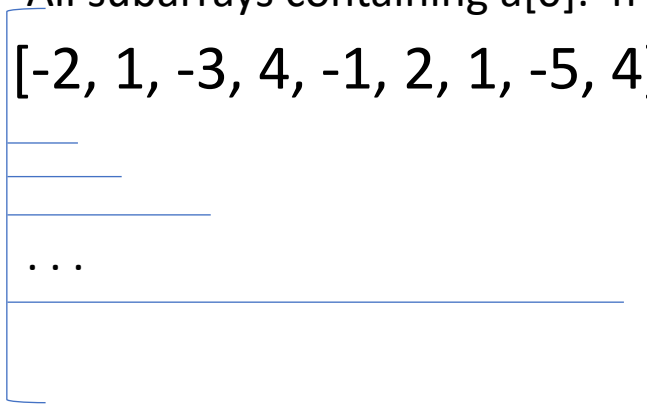
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Brute force: consider all possible subarrays

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

# Brute force: consider all possible subarrays

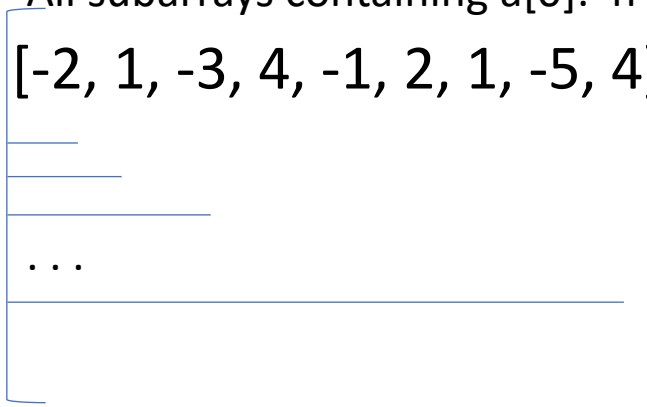
All subarrays containing a[0]: n-1



[-2, 1, -3, 4, -1, 2, 1, -5, 4]

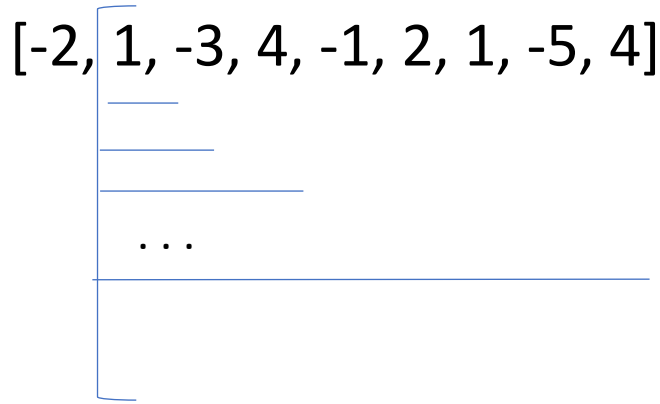
# Brute force: consider all possible subarrays

All subarrays containing a[0]: n-1




+

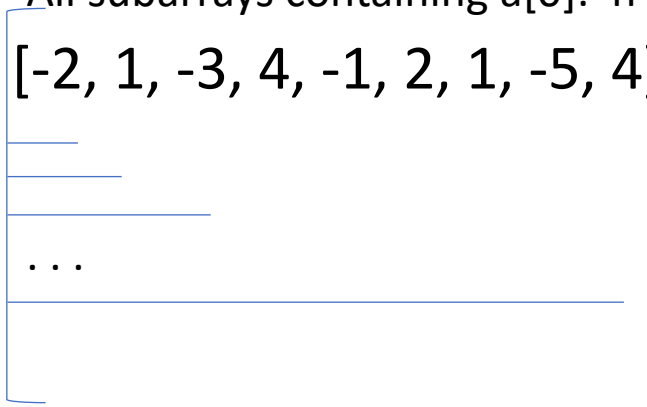
All subarrays containing a[1] but not a[0] : n-2



[-2, 1, -3, 4, -1, 2, 1, -5, 4]

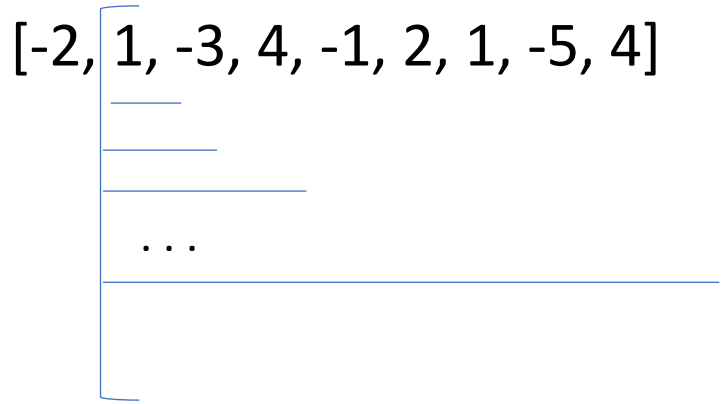
# Brute force: consider all possible subarrays

All subarrays containing a[0]: n-1




+

All subarrays containing a[1] but not a[0] : n-2

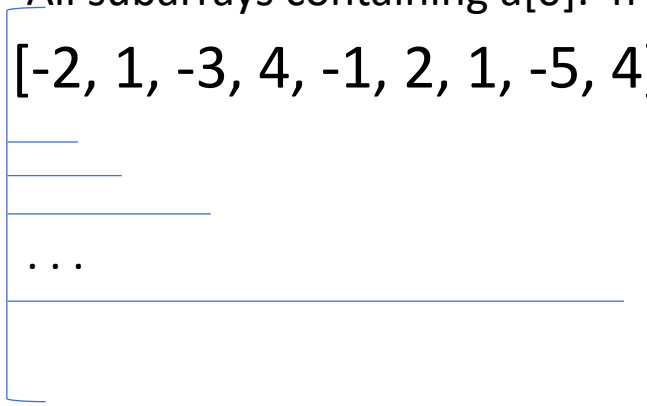



(n-1) + (n-2) + (n-3) + … + 3 + 2 + 1 = n(n-1)/2 => O(n^2)

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

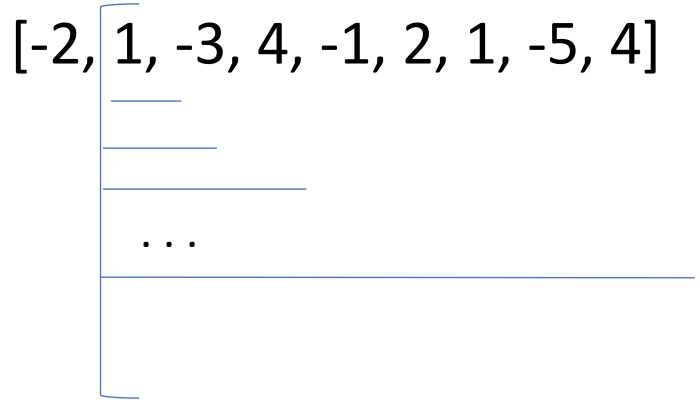
# Brute force: consider all possible subarrays

All subarrays containing a[0]: n-1




+

All subarrays containing a[1] but not a[0] : n-2




(n-1) + (n-2) + (n-3) + … + 3 + 2 + 1 = n(n-1)/2 => O(n^2)

```
for(i: 0 to n-1)
   sub_sum = 0
   for(j : i to  n-1)
       sub_sum+= a[j]
       max_sum = max(max_sum, sub_sum)
```

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

Note: all we need is the max_sum, not the subarray indices!!!

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

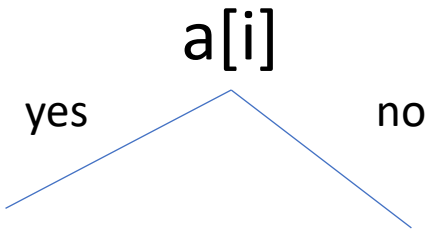
Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.

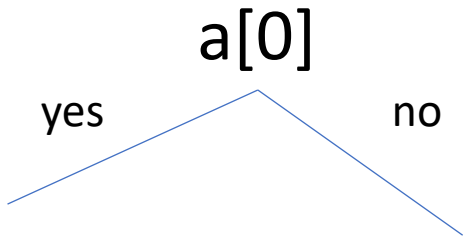


[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.

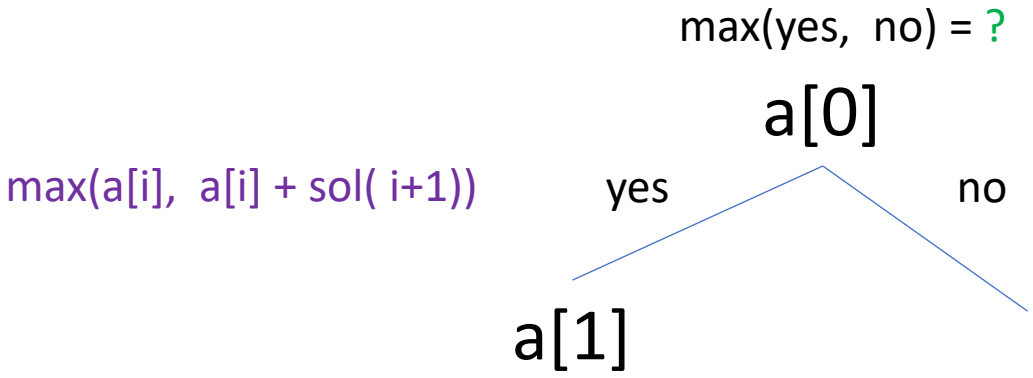
max(yes,  no) = ?



[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

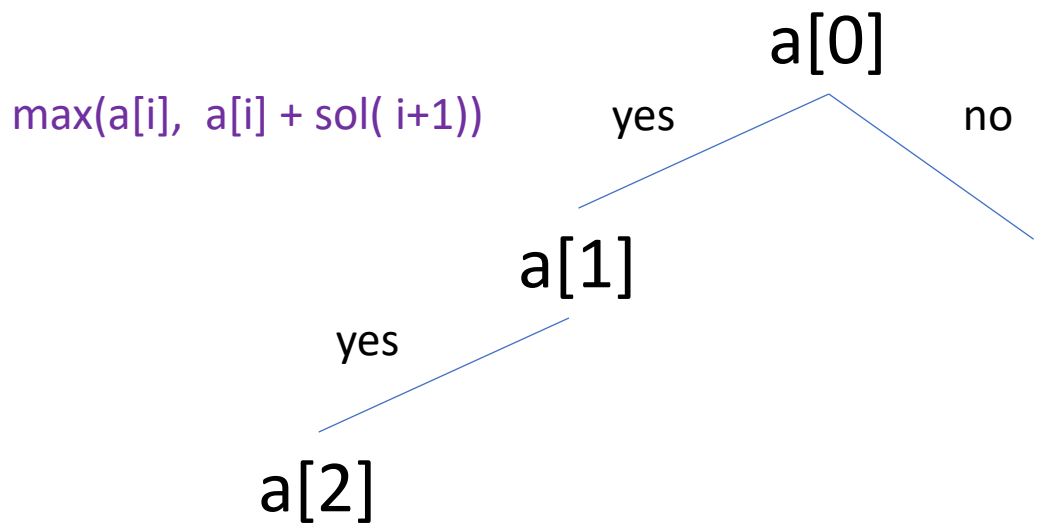
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

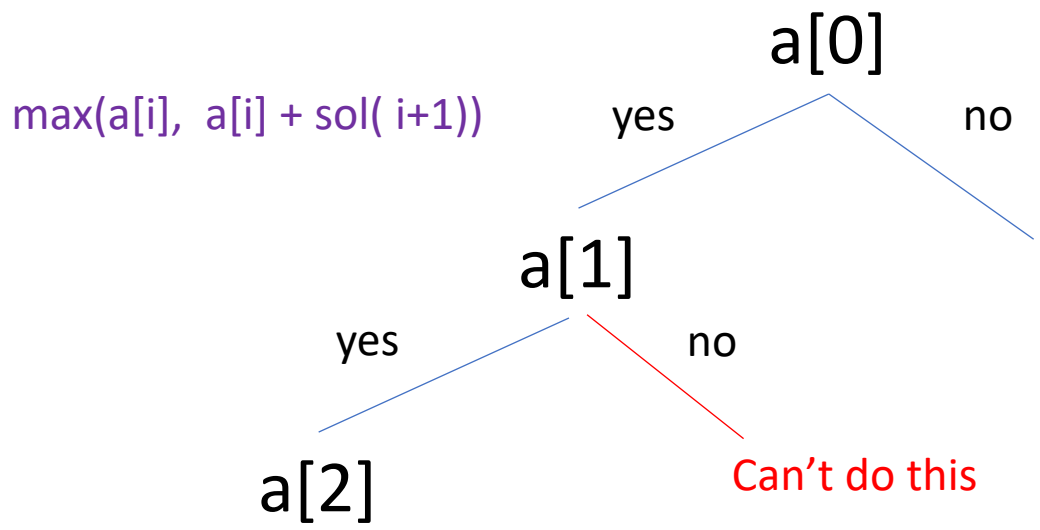
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

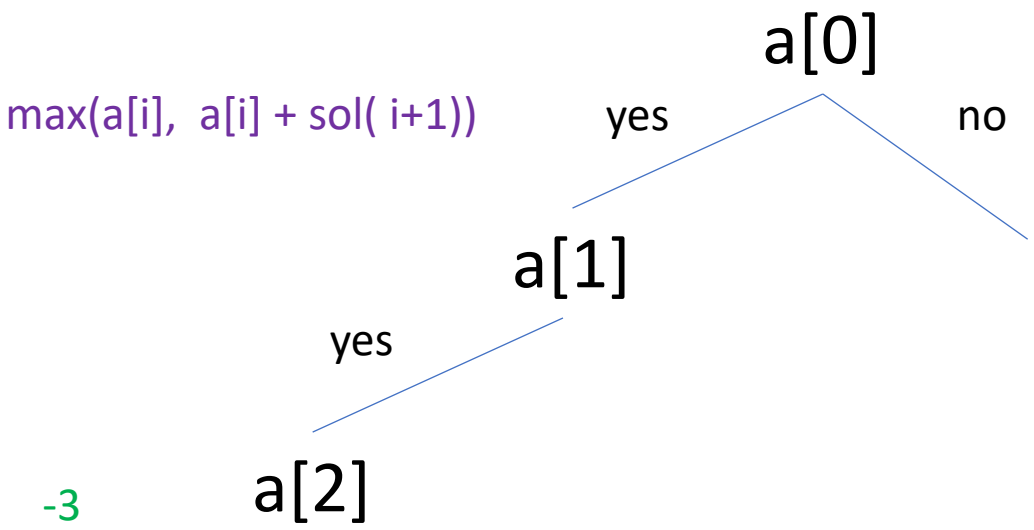
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

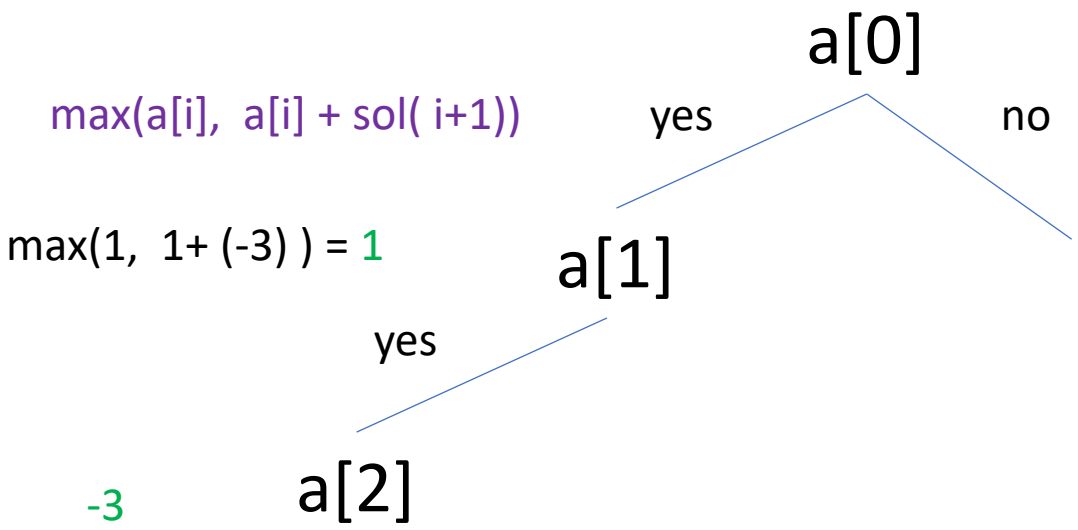
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution  from optimal solution to subproblems?

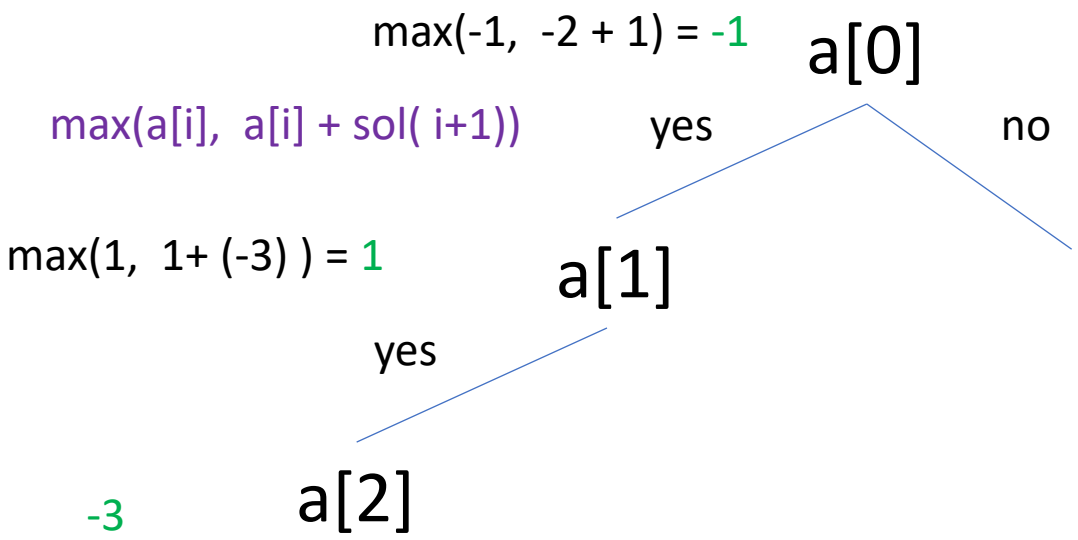
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

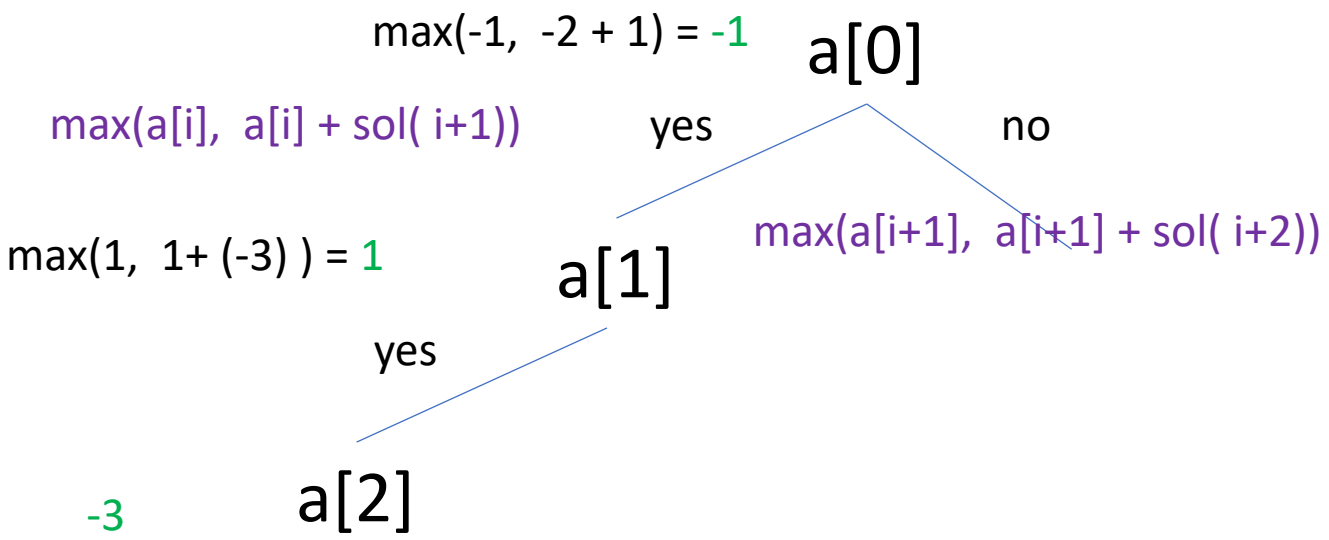
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

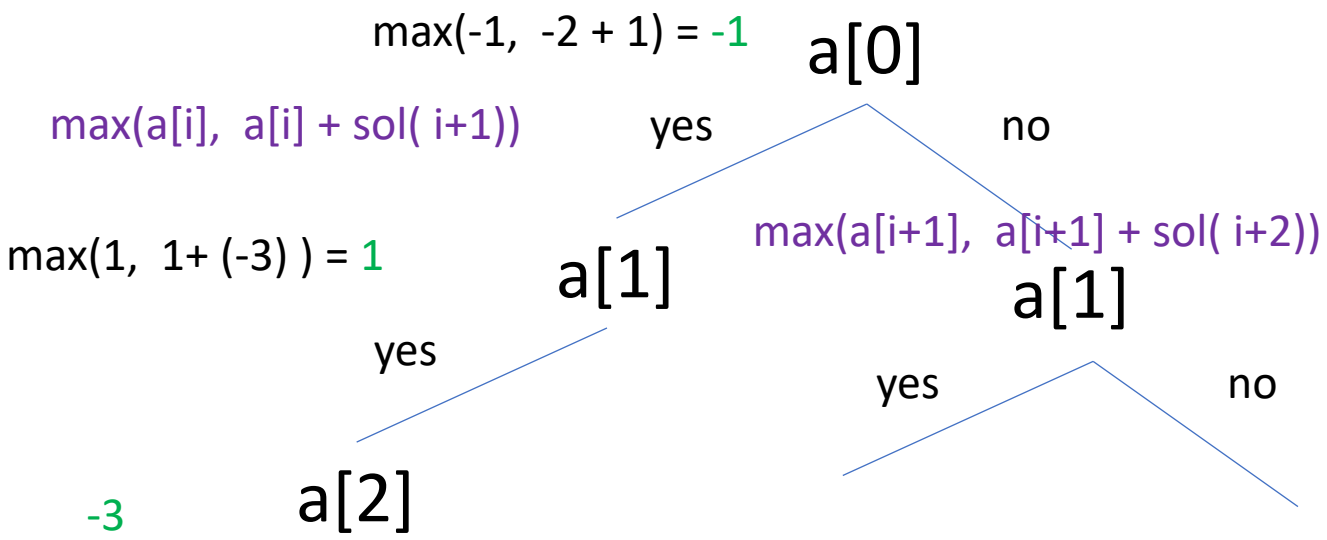
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

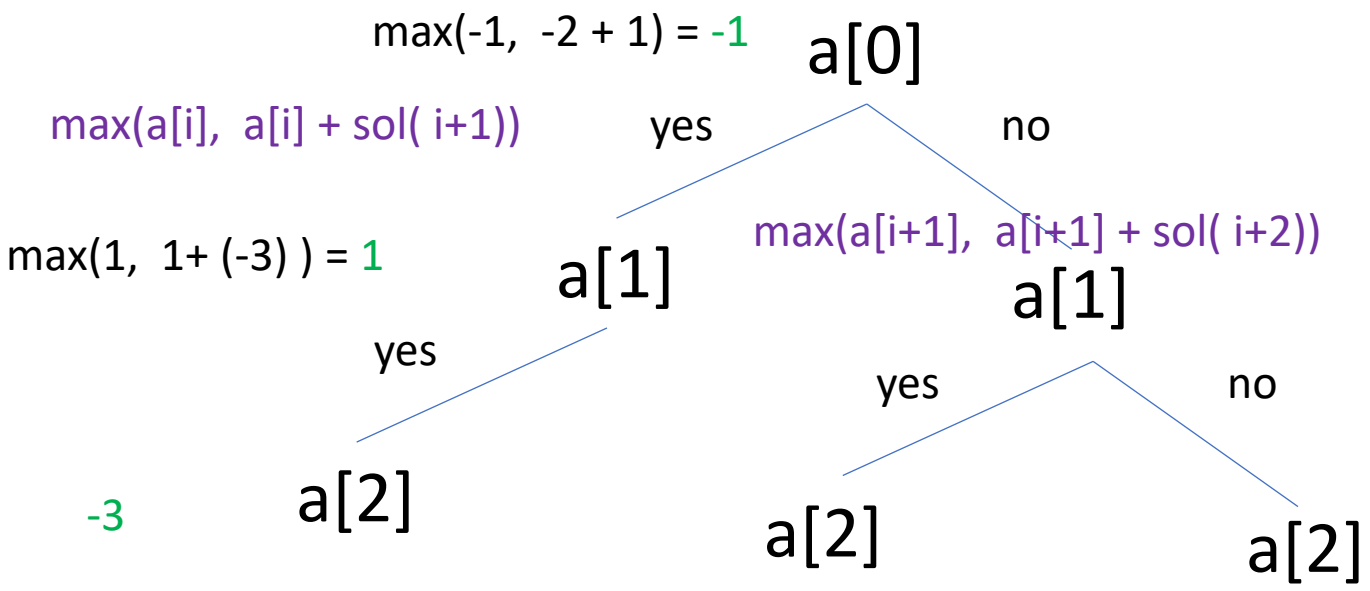
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

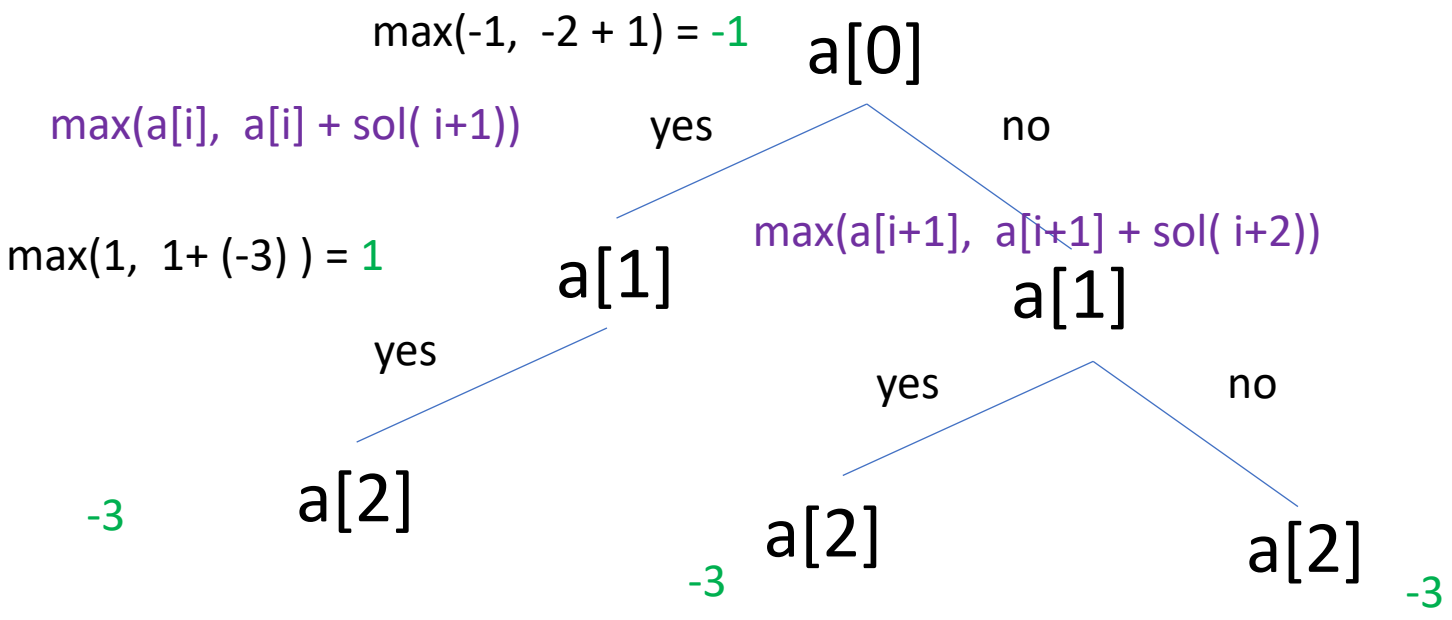
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

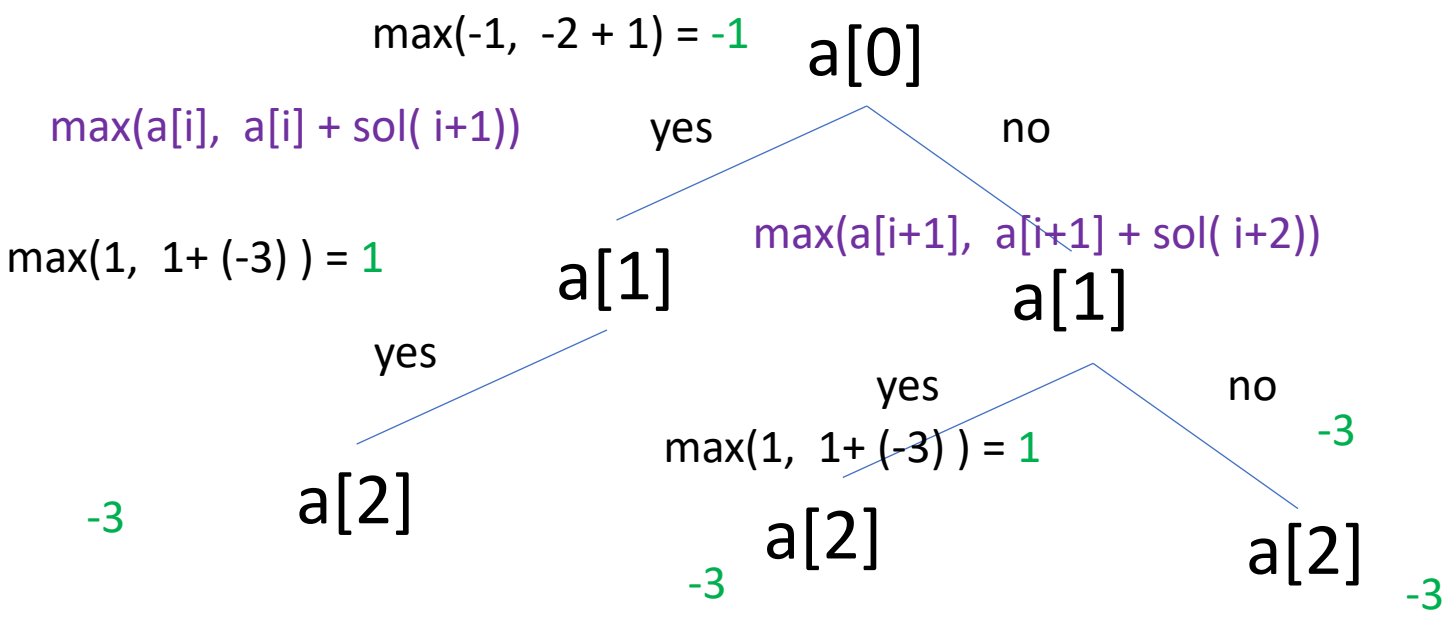
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

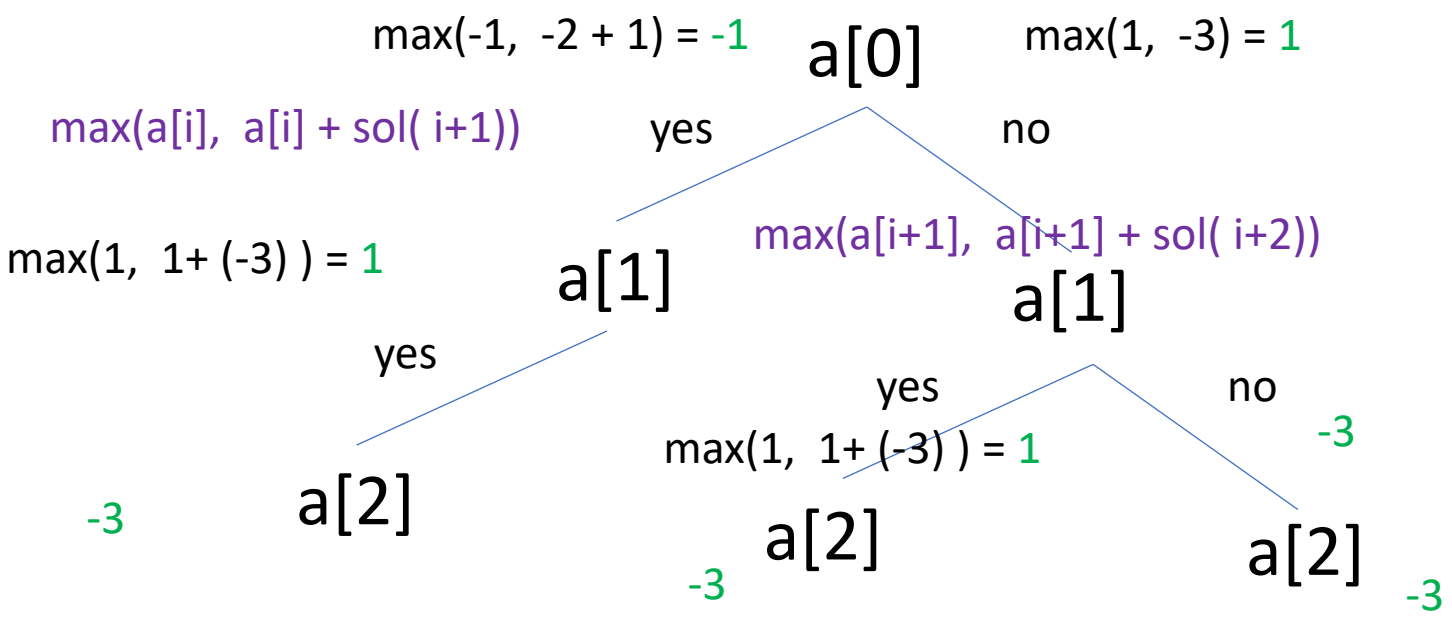
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

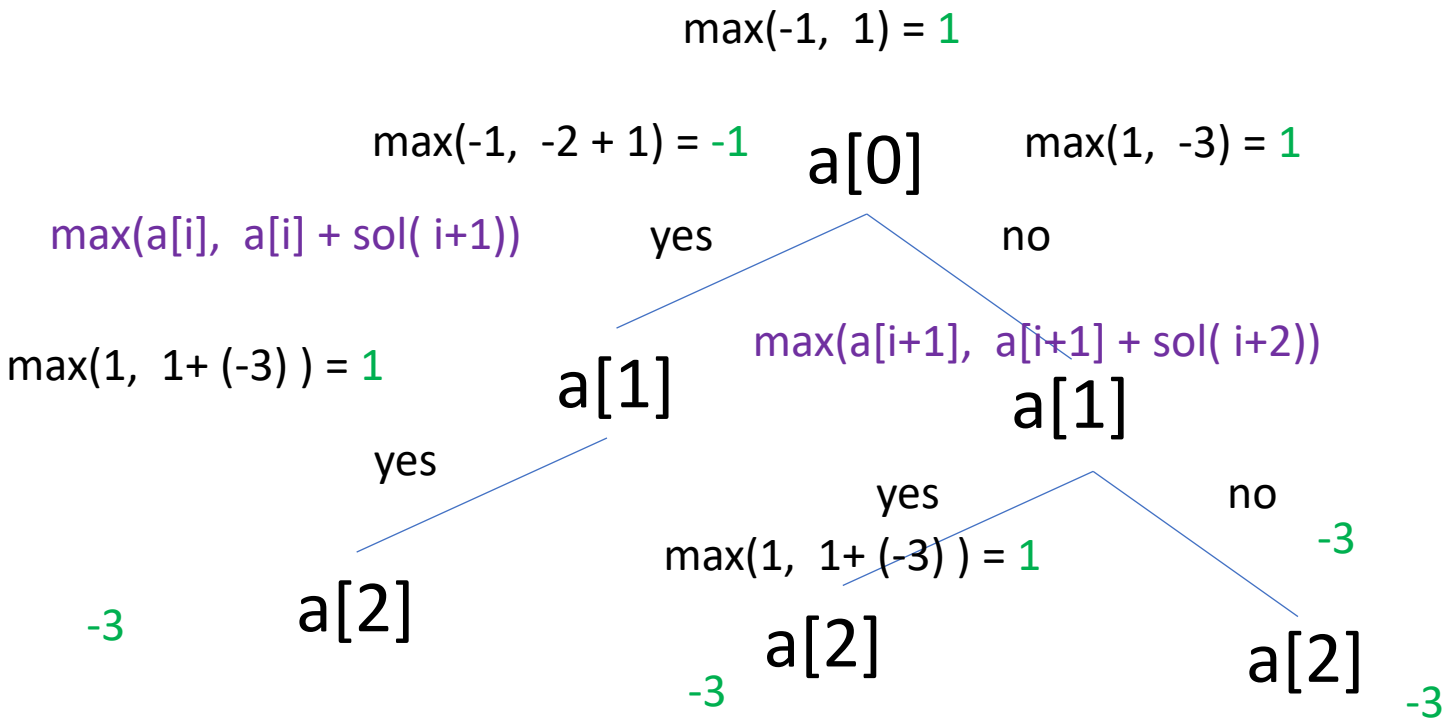
We either add a[i] in max_sum or not.



[-2, 1, -3]

# Can we define an optimal solution from optimal solution to subproblems?

We either add a[i] in max_sum or not.



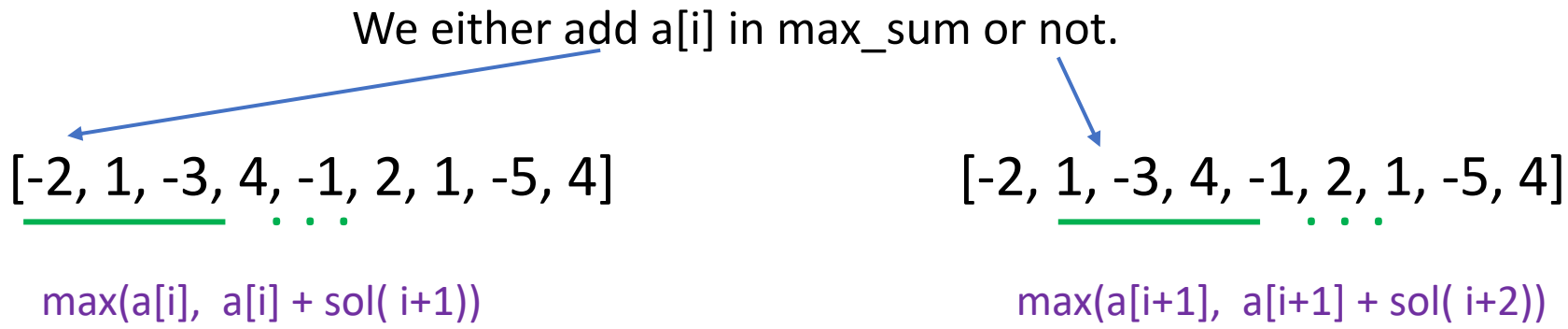
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.

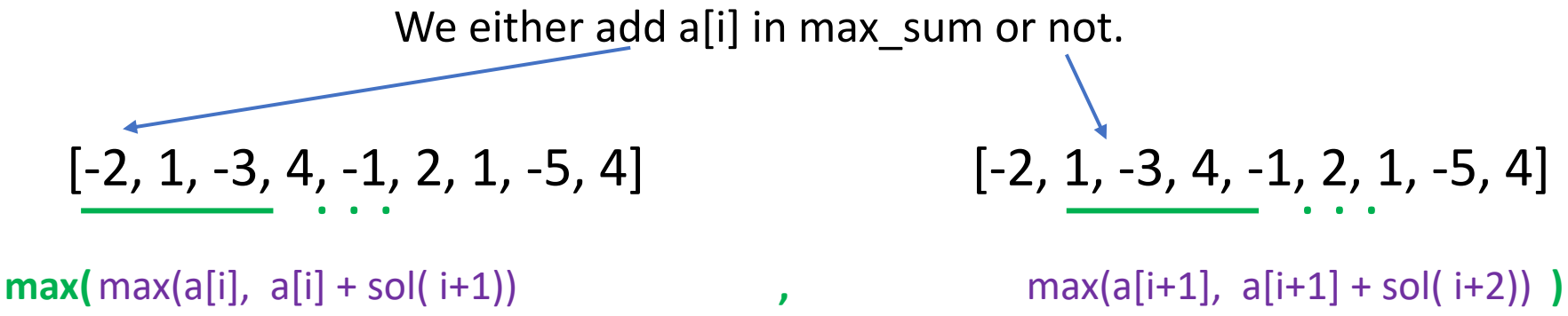
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?



[-2, 1, -3, 4, -1, 2, 1, -5, 4]

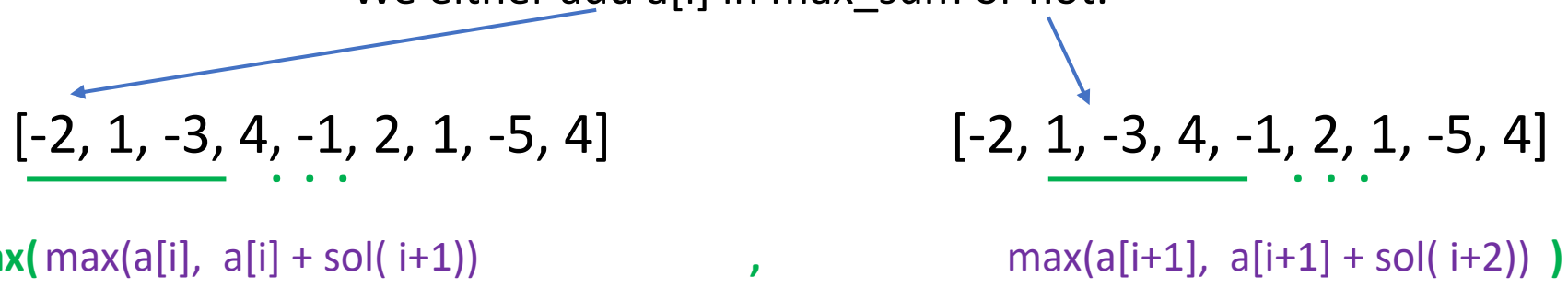
Can we define an optimal solution  from optimal solution to subproblems?



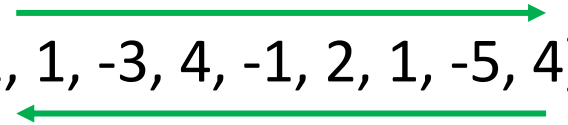
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.



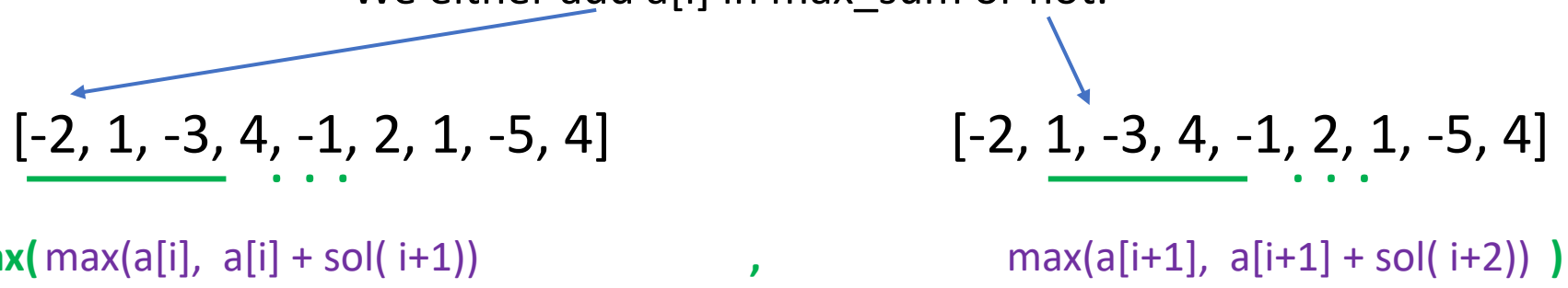

Addition is the same in both directions



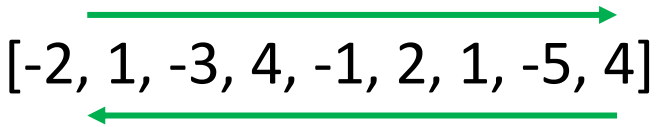
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.




Addition is the same in both directions

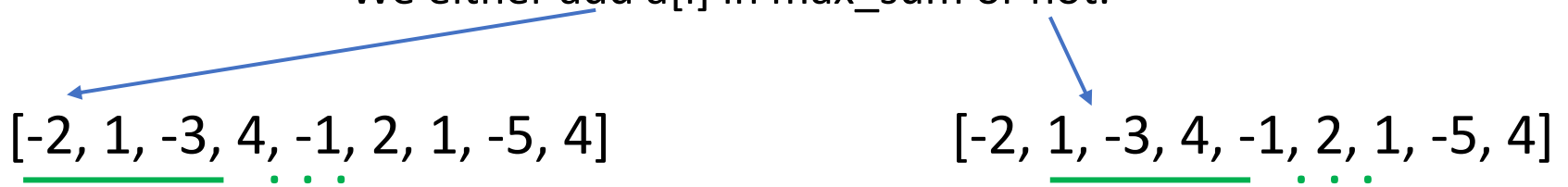



Let’s tabulate the solution to subproblems

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.




**max(** max(a[i],  a[i] + sol( i+1)) , max(a[i+1],  a[i+1] + sol( i+2)) **)**

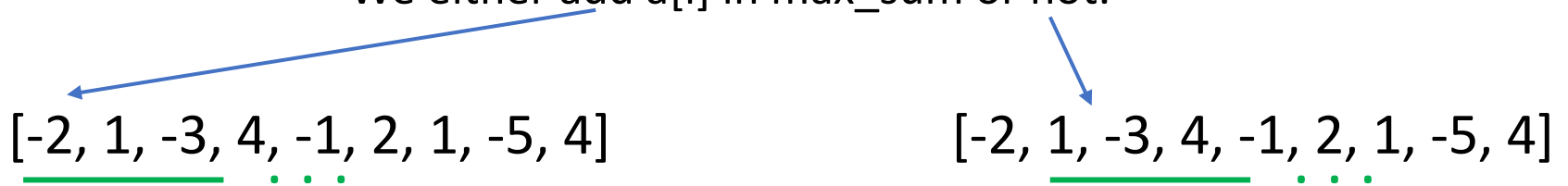
Let’s tabulate the solution to subproblems

Keep table dp (a vector) where dp[i] is the optimal solution to max_sum starting at i

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.




Let's tabulate the solution to subproblems

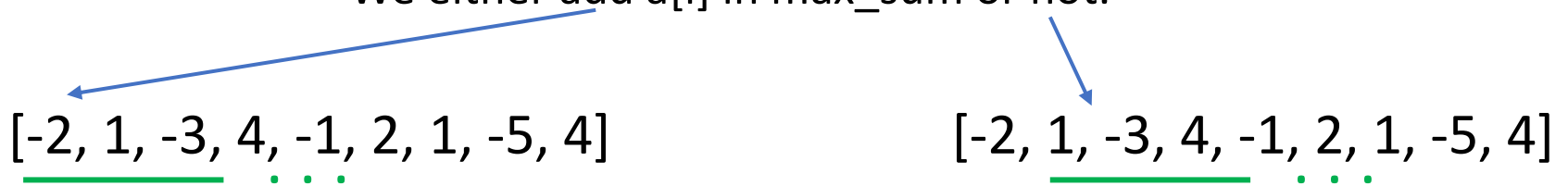
Keep table dp (a vector) where dp[i] is the optimal solution to max_sum starting at i

```
for(i: 0 to n-1)
        dp[i] = max(a[i], a[i] + dp[i-1] )
return max_element(dp)
```

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Can we define an optimal solution  from optimal solution to subproblems?

We either add a[i] in max_sum or not.




Let's tabulate the solution to subproblems

Keep table dp (a vector) where dp[i] is the optimal solution to max_sum starting at i

```
for(i: 0 to n-1)
        dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

**O(n)**

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1])
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

i=5: dp[5] = max(2, 2 + 3)
dp[-2, 1, -2, 4, 3, 5]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

i=5: dp[5] = max(2, 2 + 3)
dp[-2, 1, -2, 4, 3, 5]

i=6: dp[6] = max(1, 1 + 5)
dp[-2, 1, -2, 4, 3, 5, 6]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

i=5: dp[5] = max(2, 2 + 3)
dp[-2, 1, -2, 4, 3, 5]

i=6: dp[6] = max(1, 1 + 5)
dp[-2, 1, -2, 4, 3, 5, 6]

i=7: dp[7] = max(-5, -5 + 6)
dp[-2, 1, -2, 4, 3, 5, 6, 1]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

i=5: dp[5] = max(2, 2 + 3)
dp[-2, 1, -2, 4, 3, 5]

i=6: dp[6] = max(1, 1 + 5)
dp[-2, 1, -2, 4, 3, 5, 6]

i=7: dp[7] = max(-5, -5 + 6)
dp[-2, 1, -2, 4, 3, 5, 6, 1]

i=8: dp[8] = max(4, 4 + 1)
dp[-2, 1, -2, 4, 3, 5, 6, 1, 5]

```
for(i: 1 to n-1)
       dp[i] = max(a[i], a[i] + dp[i-1]
return max_element(dp)
```

a[-2, 1, -3, 4, -1, 2, 1, -5, 4]

dp[-2]

i=1: dp[1] = max(1, 1 + -2)
dp[-2, 1]

i=2: dp[2] = max(-3, -3 + 1)
dp[-2, 1, -2]

i=3: dp[3] = max(4, 4 + -2)
dp[-2, 1, -2, 4]

i=4: dp[4] = max(-1, -1 + 4)
dp[-2, 1, -2, 4, 3]

i=5: dp[5] = max(2, 2 + 3)
dp[-2, 1, -2, 4, 3, 5]

i=6: dp[6] = max(1, 1 + 5)
dp[-2, 1, -2, 4, 3, 5, 6]

i=7: dp[7] = max(-5, -5 + 6)
dp[-2, 1, -2, 4, 3, 5, 6, 1]

i=8: dp[8] = max(4, 4 + 1)
dp[-2, 1, -2, 4, 3, 5, 6, 1, 5]

`return max_element(dp)` = **6**