SHIFTKEY ACADEMY

Intro to Algorithmic Problem Solving

Week 3

SHIFTKEY ACADEMY

Housekeeping

- **Assignments due:** every Sunday 11:59pm
- **Soft Skill Assessment:** Post on June 27 (Fri); Due on July 6 (Sun)
- **Exam:** July 3 (Thu), 18:00-20:00
- **Exam language & make-up exam:** last call

SHIFTKEY ACADEMY

Mid-course Survey

How do you feel about the pace of the course so far?

1. **Too fast**
2. **Just right**
3. **Too slow**

SHIFTKEY ACADEMY

Mid-course Survey

Do you feel you have enough time to think during practice questions?

1. Yes, it's sufficient
2. No, I need more time
3. I would prefer less time and more questions

SHIFTKEY ACADEMY

Mid-course Survey

How helpful are the in-class examples in understanding the concepts?

1. Very helpful
2. Somewhat helpful
3. Not very helpful

SHIFTKEY ACADEMY

Mid-course Survey

Do you find the explanations clear and easy to follow?

1. Always
2. Most of the time
3. Sometimes
4. Rarely

SHIFTKEY ACADEMY

Mid-course Survey

Is the balance between theory and practice appropriate?

1. Yes, it's balanced
2. Too much theory
3. Too much practice

SHIFTKEY ACADEMY

Mid-course Survey

How do you feel about the difficulty level of the content?

1. Too easy
2. Just right
3. Too difficult

SHIFTKEY ACADEMY

Mid-course Survey

How do you feel about the amount of content provided?

1. I would prefer more content
2. The current amount is just right
3. I would prefer less content with more detailed explanation

SHIFTKEY ACADEMY

Mid-course Survey

Do you have any suggestions to improve the rest of the course?

SHIFTKEY ACADEMY

Course Overview

Week 1

Algorithm Intro
Time/Space Complexity
Data Structures
Two pointers
Prefix Sum

Week 2

Recursion
Divide and Conquer
Sorting
Binary Search

Week 3

Math Problems
Bit Manipulation
Greedy
Dynamic Programing

Week 4

Tree & Graph
BFS/DFS
Backtracking
Union-Find

SHIFTKEY ACADEMY

Math Problems

SHIFTKEY ACADEMY

69. Sqrt(x)

Solved

Easy | Topics | Companies | Hint

Given a non-negative integer `x`, return *the square root of* `x` *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

```
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.
```

Example 2:

```
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the
nearest integer, 2 is returned.
```

SHIFTKEY ACADEMY

Binary Search

Uses binary search to find the greatest integer whose square is less than or equal to x.

```python
class Solution:
    def mySqrt(self, x: int) -> int:
        low, high = 1, x
        while low <= high:
            mid = (low + high)//2
            if mid*mid == x:
                return mid
            elif mid*mid > x:
                high = mid-1
            else:
                low = mid+1
        return high
```

Time complexity: O(log n) Space complexity: O(1)

SHIFTKEY ACADEMY

Newton's Method

Applies Newton's iterative formula to quickly converge on the square root of x using integer approximation.

$$r_{n+1} = \left\lfloor \frac{r_n + \frac{x}{r_n}}{2} \right\rfloor$$

```python
class Solution:
    def mySqrt(self, x: int) -> int:
        r = x
        while r*r > x:
            r = (r + x//r)//2
        return r
```

Time complexity: O(log n)
* faster than binary search

Space complexity: O(log n)

SHIFTKEY ACADEMY

50. Pow(x, n)

Medium | Topics | Companies

Implement pow(x, n), which calculates `x` raised to the power `n` (i.e., x^n).

Example 1:

```
Input: x = 2.00000, n = 10
Output: 1024.00000
```

Example 2:

```
Input: x = 2.10000, n = 3
Output: 9.26100
```

Example 3:

```
Input: x = 2.00000, n = -2
Output: 0.25000
Explanation: 2^-2 = 1/2^2 = 1/4 = 0.25
```

SHIFTKEY ACADEMY

Exponentiation by Squaring

Recursively computes x^n using divide-and-conquer by halving the exponent and squaring the base, which reduces time complexity to O(log n).

```python
class Solution:
    def myPow(self, x: float, n: int) -> float:
        if not n:
            return 1
        if n < 0:
            return 1 / self.myPow(x, -n)
        if n % 2:
            return x * self.myPow(x, n-1)
        return self.myPow(x*x, n//2)
```

Time complexity: O(log n) Space complexity: O(log n)

SHIFTKEY ACADEMY

Bit Manipulation

SHIFTKEY ACADEMY

Bit Manipulation

- A technique that works directly on the binary representation of integers using bitwise operators.
- **Bitwise operators:**

Operator	Name	Example	Explanation
&	Bitwise AND	5 & 3 → 1	0101 & 0011 = 0001 → Only 1 if both bits are 1
|	Bitwise OR	5 | 3 → 7	0101 | 0011 = 0111 → 1 if either bit is 1
^	Bitwise XOR	5 ^ 3 → 6	0101 ^ 0011 = 0110 → 1 if bits are different
<<	Shift Left	3 << 1 → 6	0011 << 1 = 0110 → Shifts bits left (multiply by 2)
>>	Shift Right	8 >> 2 → 2	1000 >> 2 = 0010 → Shifts bits right (divide by 4)
~	Bitwise NOT (Complement)	~5 → -6	~0101 = 1010 → Inverts all bits (result is -x - 1)

SHIFTKEY ACADEMY

191. Number of 1 Bits

Easy | Topics | Companies

Given a positive integer `n`, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Example 1:

> **Input:** `n = 11`
>
> **Output:** 3
>
> **Explanation:**
>
> The input binary string **1011** has a total of three set bits.

Example 2:

> **Input:** `n = 128`
>
> **Output:** 1
>
> **Explanation:**
>
> The input binary string **10000000** has a total of one set bit.

Example 3:

> **Input:** `n = 2147483645`
>
> **Output:** 30
>
> **Explanation:**
>
> The input binary string **11111111111111111111111111111101** has a total of thirty set bits.

Set Bit

A set bit refers to a bit in the binary representation of a number that has a value of `1`.

Constraints:

- $1 <= n <= 2^{31} - 1$

SHIFTKEY ACADEMY

Bit Manipulation

Counts the number of 1s in the binary representation of a 32-bit integer by checking each bit with a right shift and bitwise AND.

```python
class Solution:
    def hammingWeight(self, n: int) -> int:
        count = 0
        for i in range(32):
            if (n>>i & 1) == 1:
                count += 1
        return count
```

Time complexity: O(32) = O(1) Space complexity: O(1)

SHIFTKEY ACADEMY

Brian Kernighan's Algorithm

Repeatedly removes the lowest set bit from n, counting how many times it can do so until n becomes zero.

```python
class Solution:
    def hammingWeight(self, n: int) -> int:
        count = 0
        while n:
            n &= n - 1
            count += 1
        return count
```

Time complexity: O(k)

*k is the number of 1s

Space complexity: O(1)

SHIFTKEY ACADEMY

136. Single Number

Easy | Topics | Companies | Hint

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

> **Input:** `nums = [2,2,1]`
>
> **Output:** `1`

Example 2:

> **Input:** `nums = [4,1,2,1,2]`
>
> **Output:** `4`

Example 3:

> **Input:** `nums = [1]`
>
> **Output:** `1`

SHIFTKEY ACADEMY

Bit Manipulation

Uses the bitwise XOR operation to cancel out all numbers that appear twice, leaving only the unique number that appears once.

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        ans = 0
        for i in nums:
            ans ^= i
        return ans
```

Time complexity: O(n) Space complexity: O(1)

SHIFTKEY ACADEMY

Greedy Algorithm

SHIFTKEY ACADEMY

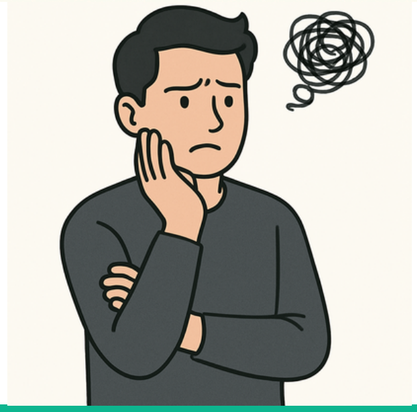
Greedy Algorithm

- A greedy algorithm makes a sequence of choices, each of which looks best at the moment, aiming for a local optimum **in hopes of** reaching a global optimum.
- **Key Characteristics:**
  - Makes locally optimal choices at each step (One strategy for the whole process)
  - No backtracking or reconsideration

SHIFTKEY ACADEMY

Activity Selection

- **Shiftkey Labs** is hosting 5 events on the same day. Each event has a **start time** and an **end time**, but some events overlap in schedule.
- Your goal is to attend as many non-overlapping events as possible – that is, you want to **maximize the number of events you can attend** without time conflicts.
- You can only attend **one event at a time**, and you must be present for the entire duration of any event you choose to attend.



SHIFTKEY ACADEMY

Activity Selection

- Select activity with the shortest duration first ✗

- Select the activity that starts earliest ✗

- Select the activity with the least number of conflicts ✗

SHIFTKEY ACADEMY

Activity Selection

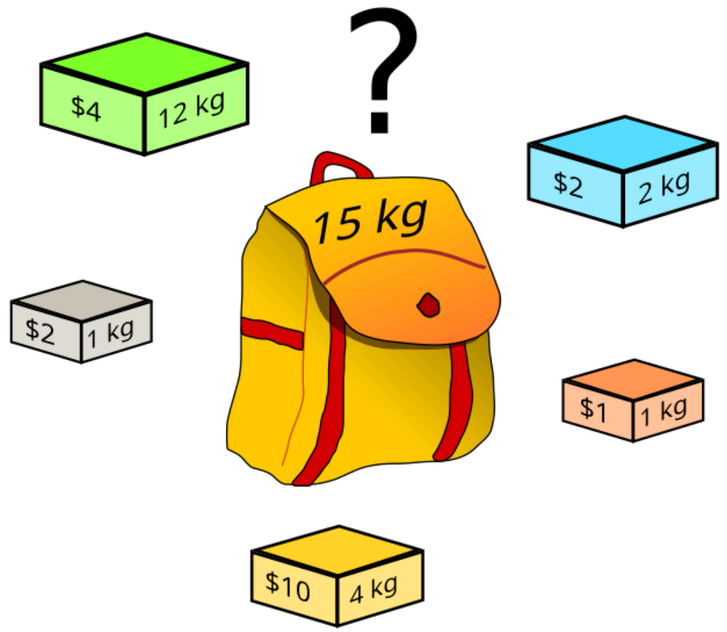
- **Select the activity that finishes earliest among the remaining compatible activities ✅**

- **Why it works:**
  - **Greedy-choice**: A globally optimal solution can be arrived at by choosing the local optimal solution.
  - **Optimal substructure**: The problem can be broken into subproblems that are themselves activity selection problems.

SHIFTKEY ACADEMY

Fractional Knapsack Problem

- You’re preparing some snacks for a Shiftkey Labs' event. You have a limited backpack with **a weight capacity of 15 kg**, and there are several boxes of snacks you can choose from.
- Each box has a total **weight** and a **value**.
- You want to **maximize the total value** of snacks you bring – but here's the catch:
  - You’re allowed to take partial boxes. That means you can open a box and only pack part of it, proportionally gaining part of its value.



SHIFTKEY ACADEMY

Fractional Knapsack Problem

- **Always pack the item with the highest value-to-weight ratio first, and take as much of it as possible.✅**

$$\max_i \left(\frac{v_i}{w_i} \right)$$

- **What if the items are not fractional?**
  - Greedy may make an early choice that blocks a better combination later.
    → In this case, you need **dynamic programming**

SHIFTKEY ACADEMY

Dynamic Programming

SHIFTKEY ACADEMY

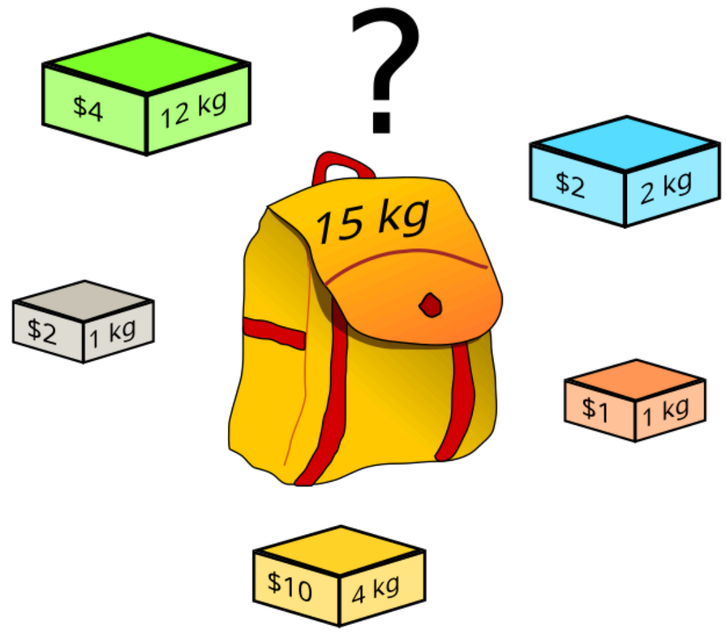
Dynamic Programming

- Dynamic Programming solves problems by breaking them down into overlapping subproblems, storing results to avoid redundant work.
- **Key Characteristics:**
  - Solves each subproblem once, then reuses the answer
  - Requires optimal substructure and overlapping subproblems
  - Usually involves recursion + memoization or bottom-up tabulation

SHIFTKEY ACADEMY

0-1 Knapsack Problem

- In this case, you cannot divide the items – you must either take the entire item or leave it.
- Your goal remains the same: to **maximize the total value** without exceeding the weight capacity of the knapsack.



SHIFTKEY ACADEMY

0-1 Knapsack Problem

- Let **dp[i][w]** represent the maximum value achievable by using the first **i** items with a knapsack of capacity **W**.
- **Base case:**

$$dp[0][w] = 0 \quad \text{for all } w$$

$$dp[i][0] = 0 \quad \text{for all } i$$

- **Recurrence relation:**
  - If the current item's weight $\mathbf{w_i}$ is greater than the capacity **W**:

$$dp[i][w] = dp[i-1][w]$$

  - Otherwise:

Don't take the item

Take the item

$$dp[i][w] = \max\left(dp[i-1][w],\ dp[i-1][w-w_i] + v_i\right)$$

SHIFTKEY ACADEMY

0-1 Knapsack Problem

```python
capacity = 15
n = len(items)

# Initialize DP table
dp = [[0 for w in range(capacity + 1)] for i in range(n + 1)]

# Fill DP table
for i in range(1, n + 1):
    value, weight = items[i - 1]
    for w in range(capacity + 1):
        if weight > w:
            dp[i][w] = dp[i - 1][w]
        else:
            dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weight] + value)

# Final result
print("Maximum value:", dp[n][capacity])
```

Item	Value	Weight
1	4	12
2	2	2
3	2	1
4	1	1
5	10	4

Time complexity: $O(n \cdot W)$ Space complexity: $O(n \cdot W)$

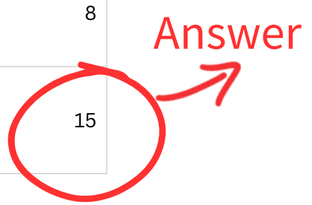
*filling the n x W table

SHIFTKEY ACADEMY

0-1 Knapsack Problem

Table dp

	W=0	W=1	W=2	W=3	W=4	W=5	W=6	W=7	W=8	W=9	W=10	W=11	W=12	W=13	W=14	W=15
Item 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Item 1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
Item 2	0	0	2	2	2	2	2	2	2	2	2	2	4	4	6	6
Item 3	0	2	2	4	4	4	4	4	4	4	4	4	4	6	6	8
Item 4	0	2	3	4	5	5	5	5	5	5	5	5	5	6	7	8
Item 5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15



SHIFTKEY ACADEMY

0-1 Knapsack Problem

```
for w in range(capacity + 1):
    if weight > w:
        dp[i][w] = dp[i - 1][w]
    else:
        dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weight] + value)
```

```
for w in range(capacity + 1):
    if weight > w:
        dp[i][w] = dp[i-1][w]        # cannot take item i
    else:
        without = dp[i-1][w]
        with_it = dp[i-1][w-weight] + value
        if with_it > without:        # choose item i
            dp[i][w] = with_it
            S[i][w]  = 1
        else:                        # skip item i
            dp[i][w] = without
            S[i][w]  = 0
```

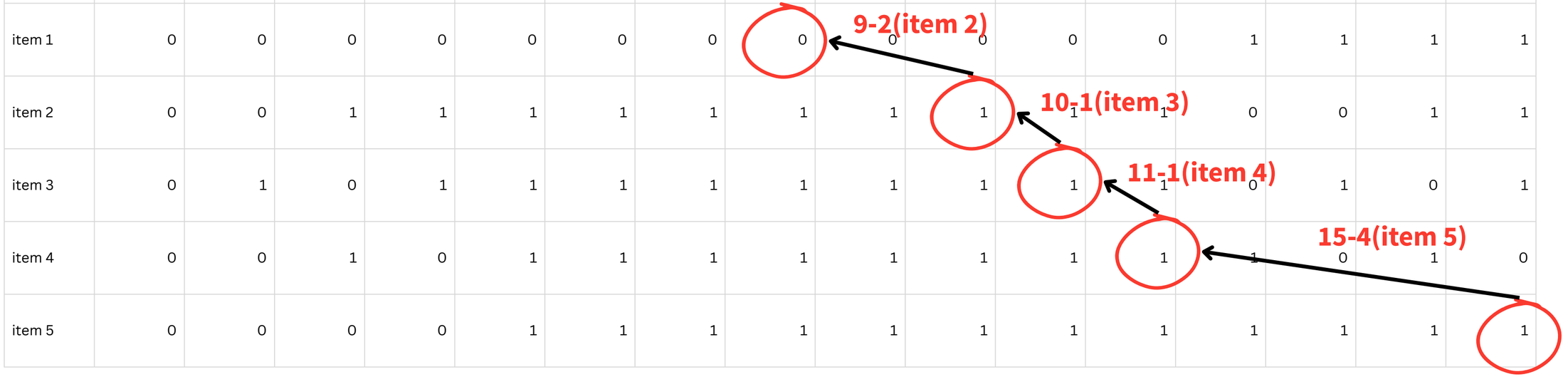
- To keep tracking our choice.
- The table S[i][w] stores a 1 when item i is chosen for capacity w, and 0 otherwise.

SHIFTKEY ACADEMY

0-1 Knapsack Problem

Table S

	W=0	W=1	W=2	W=3	W=4	W=5	W=6	W=7	W=8	W=9	W=10	W=11	W=12	W=13	W=14	W=15
item 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
item 1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
item 2	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1
item 3	0	1	0	1	1	1	1	1	1	1	1	1	0	1	0	1
item 4	0	0	1	0	1	1	1	1	1	1	1	1	1	0	1	0
item 5	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1



Answer: select item 2, item 3, item 4, item 5

SHIFTKEY ACADEMY

Fibonacci (Top-Down)

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

```python
def fib(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        return 0
    if n == 1:
        return 1
    memo[n] = fib(n - 1, memo) + fib(n - 2, memo)
    return memo[n]
```

Time complexity: O(n)
Space complexity: O(n)
(at most n elements in memory)

SHIFTKEY ACADEMY

Fibonacci (Bottom-Up)

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n \geq 2 \end{cases}$$

```python
def fib(n):
    if n <= 1:
        return n
    dp = [0, 1]
    for i in range(2, n+1):
        dp.append(dp[i-1] + dp[i-2])
    return dp[n]
```

Time complexity: O(n)
Space complexity: O(n)
Optimization?

Only keep last two values → O(1) space

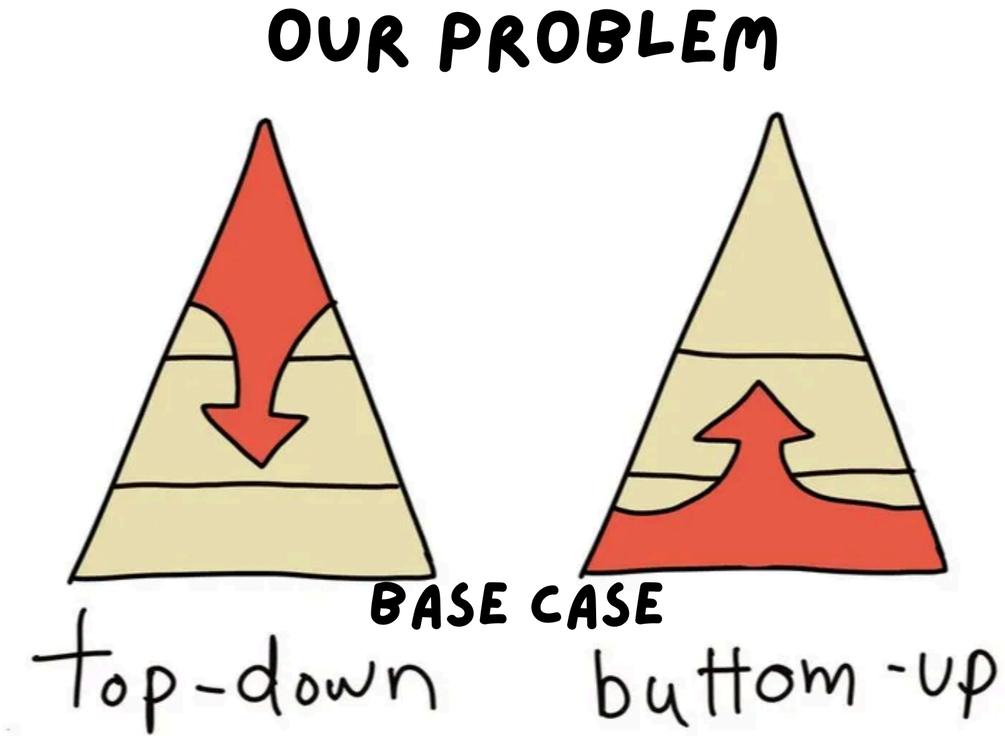
SHIFTKEY ACADEMY

Top-Down vs. Bottom-Up

- **Top-Down:**
  - Start from the original problem and recursively break it down into subproblems. Often combined with memoization to avoid redundant work.
- **Bottom-Up:**
  - Start from the base cases and iteratively build up to the final answer. This is typically implemented using loops and often called Dynamic Programming (DP).

SHIFTKEY ACADEMY

Top-Down vs. Bottom-Up

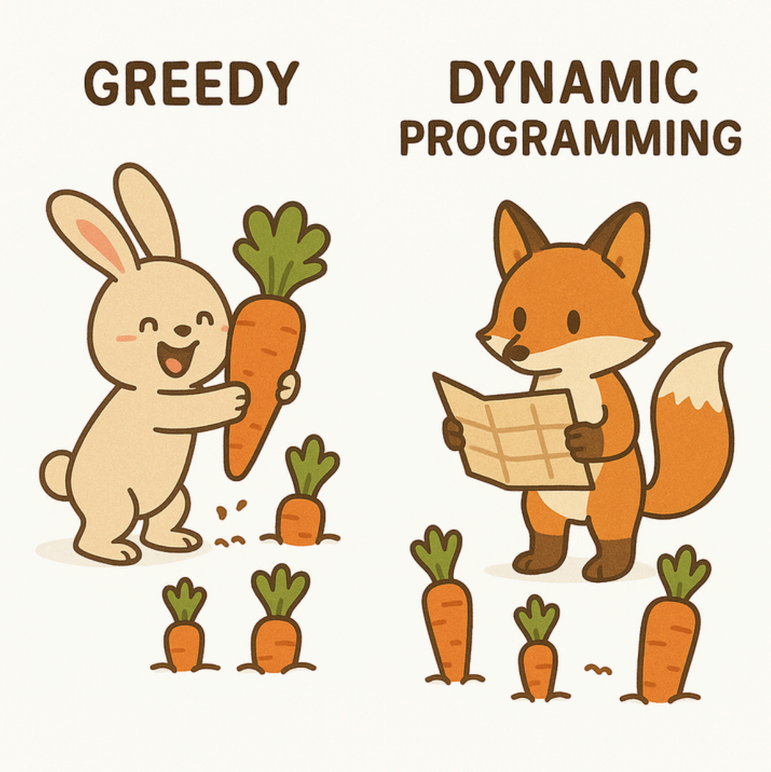


Q: Are Top-Down and Bottom-Up Opposites?

A: **Not exactly.** Even though recursive Top-Down algorithms call down, they solve up.

SHIFTKEY ACADEMY

Greedy VS. Dynamic Programming



	Greedy Algorithm	Dynamic Programming
Core Idea	Makes the best local choice at each step	Breaks the problem into overlapping subproblems, solves each optimally, and combines them for the global solution
Decision Strategy	Irrevocable choices; only considers the current step	Considers all possible subproblem combinations to ensure the best overall decision
Overlapping Subproblems	Typically none or minimal	Yes; subproblems recur and benefit from reuse
Guarantee of Optimality	Not always; may lead to suboptimal solutions	Always guarantees optimal solution
Time Complexity	Often faster, usually O(n) or O(n log n)	Usually slower, often $O(n^2)$ or worse
Space Complexity	Low; often O(1) or O(n)	Higher; requires additional memory for subproblem solutions

SHIFTKEY ACADEMY

Practice Questions

SHIFTKEY ACADEMY

55. Jump Game

Solved

Medium | Topics | Companies

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` *if you can reach the last index, or* `false` *otherwise.*

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.
```

SHIFTKEY ACADEMY

Greedy

Greedily tracks the farthest index reachable at each step and returns False if the current position is beyond that reach.

```python
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        farthest = 0
        n = len(nums)
        for i in range(n):
            if i > farthest:
                return False
            farthest = max(farthest, i + nums[i])
        return farthest >= n - 1
```

Time complexity: O(n) Space complexity: O(1)

SHIFTKEY ACADEMY

121. Best Time to Buy and Sell Stock

Solved

Easy | Topics | Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

SHIFTKEY ACADEMY

Dynamic Programming

Finds the maximum profit from a single stock buy-sell transaction by tracking the lowest price seen so far and calculating the best profit at each step.

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        lsf = inf
        op = 0
        for i in range(len(prices)):
            lsf = min(lsf, prices[i])
            op = max(op, prices[i]-lsf)
        return op
```

Time complexity: O(n) Space complexity: O(1)

SHIFTKEY ACADEMY

62. Unique Paths

Solved

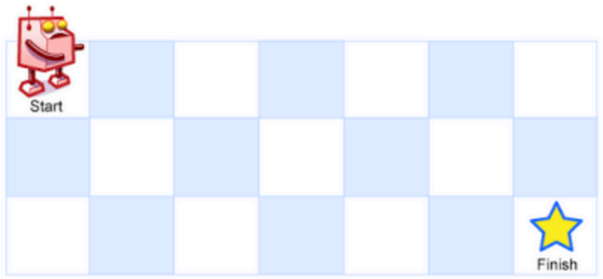
Medium | Topics | Companies

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to `2 * 10^9`.

Example 1:




```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

SHIFTKEY ACADEMY

Dynamic Programming(2D)

Use a 2D array dp[i][j] to store the number of unique paths to reach cell (i, j).

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[1]*n for _ in range(m)]
        for i in range(1,m):
            for j in range(1,n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1]
        return dp[m-1][n-1]
```

Time complexity: O(n*m) Space complexity: O(n*m)

SHIFTKEY ACADEMY

Dynamic Programming(1D)

Reduce space to 1D since each row only depends on the previous row.

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [1]*n
        for i in range(1,m):
            for j in range(1,n):
                dp[j] += dp[j-1]
        return dp[-1]
```

Time complexity: O(n*m) Space complexity: O(n)

SHIFTKEY ACADEMY

Quiz

SHIFTKEY ACADEMY

Reference

https://github.com/changgyhub/leetcode_101

https://github.com/halfrost/LeetCode-Go

https://github.com/krahets/hello-algo

SHIFTKEY ACADEMY

THANK YOU TO OUR PARTNERS

cognizant®

NOVA SCOTIA

DALHOUSIE UNIVERSITY | COMPUTER SCIENCE

SHIFTKEY ACADEMY

We want to hear from you!

Scan the QR Code to provide your feedback