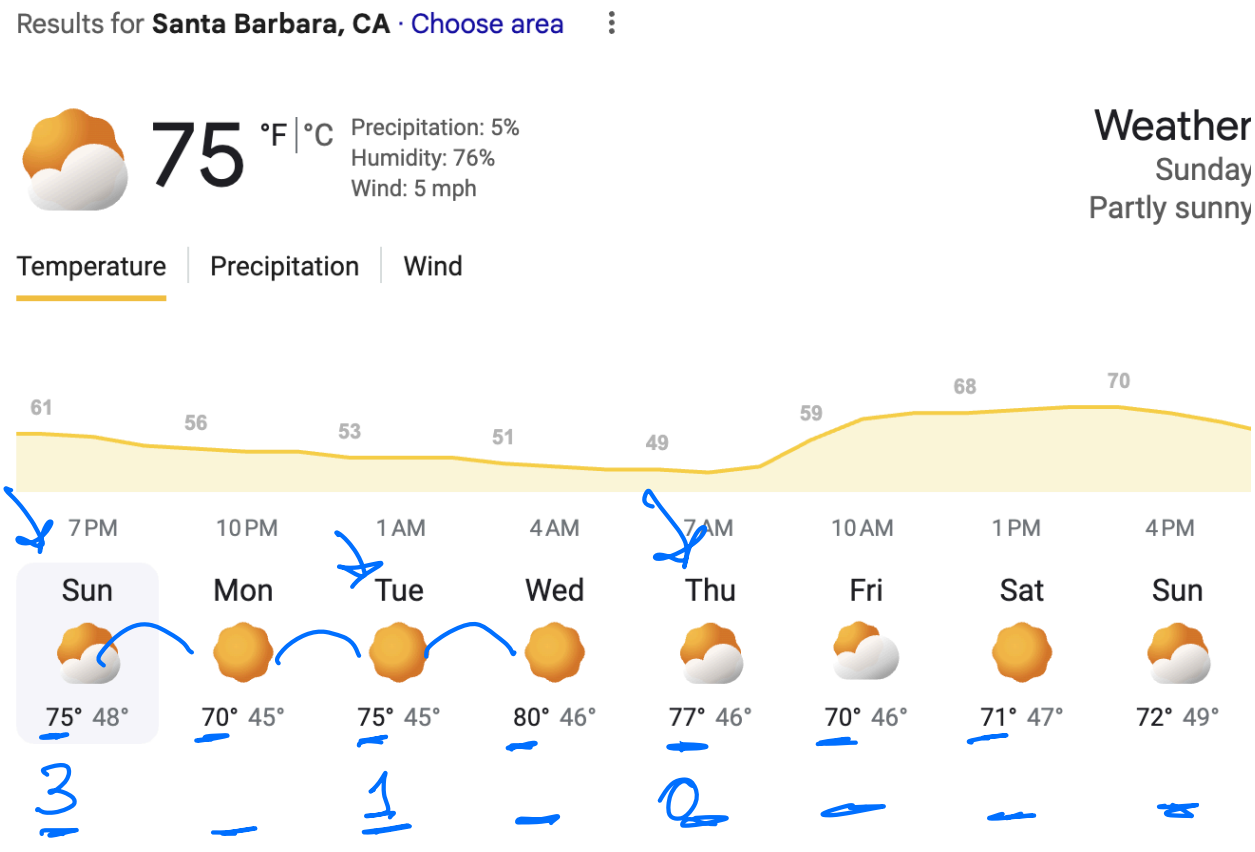
Stacks and Santa Barbara Weather Puzzle

A stack manages keys using the principle of Last in First Out (LIFO) via four operations, each O(1): 1. push(value) 2. pop() 3. top() 4. empty()

A stack is useful for keeping track of history information where computation only depends on the most recent information !!

Objective: Analyze the complexity of a naive solution for the Daily Temperatures problem by determining its complexity. Next, optimize using a stack-based solution. Explore time and space complexity tradeoff.

Problem: Given an array of daily temperatures, return an array `answer` where `answer[i]` is the number of days after day `i` until a *warmer* temperature occurs, or 0 if none exists. Use Santa Barbara's forecast:



Leetcode (medium) daily temperatures:
https://leetcode.com/problems/daily-temperatures/

Approach: For every day (i), scan the input vector from i+1 to n-1 until a warmer day is reached.

Temp:	75	70	75	80	77	70	71	72
Day:	0	1	2	3	4	5	6	7

Part 1: Naive Solution - Turn the problem's definition into an algorithm.

Approach: For each i, look to future days to find a warmer day

Ans: [3, 1, 1, 0, 0, 1, 1, 0]

Fill in the blanks to complete pseudocode:

How many times does this statement run over all?

```
None
Initialize answer = [0] * n
For each day i from n-1 down to 0:
    For each day j from i+1 to n-1:
        If temperatures[j] ____ temperatures[i]:
            answer[i] = ____
            break
return answer
```

Part 2: Complexity Analysis of Naive Solution (Discuss with your peer)

1. How many comparisons did you make to get the answer for day 7 for the given input? 0
2. How many comparisons did you make to get the answer for day 0 for the given input? 7
3. Write an 8-day temperature input vector that incurs the worst-case running time

Any non-increasing sequence → [75, 75, 75, 75, 74, 70, 69, 53]

4. What is the worst-case time and space complexity expressed in Big-O $S(n) = O(1)$

i	No. of comparisons
n-1	0
n-2	1
n-3	2
⋮	⋮
0	(n-1)

No. of comparisons $= (n-1) - (i+1) + 1$
$= (n-i-1)$

$T(n) = 0 + 1 + 2 + \cdots + (n-1)$

$= \frac{(n-1)(n-1+1)}{2}$

$= \frac{n(n-1)}{2} = O(n^2)$

(10 mins) Discuss with your neighbor:

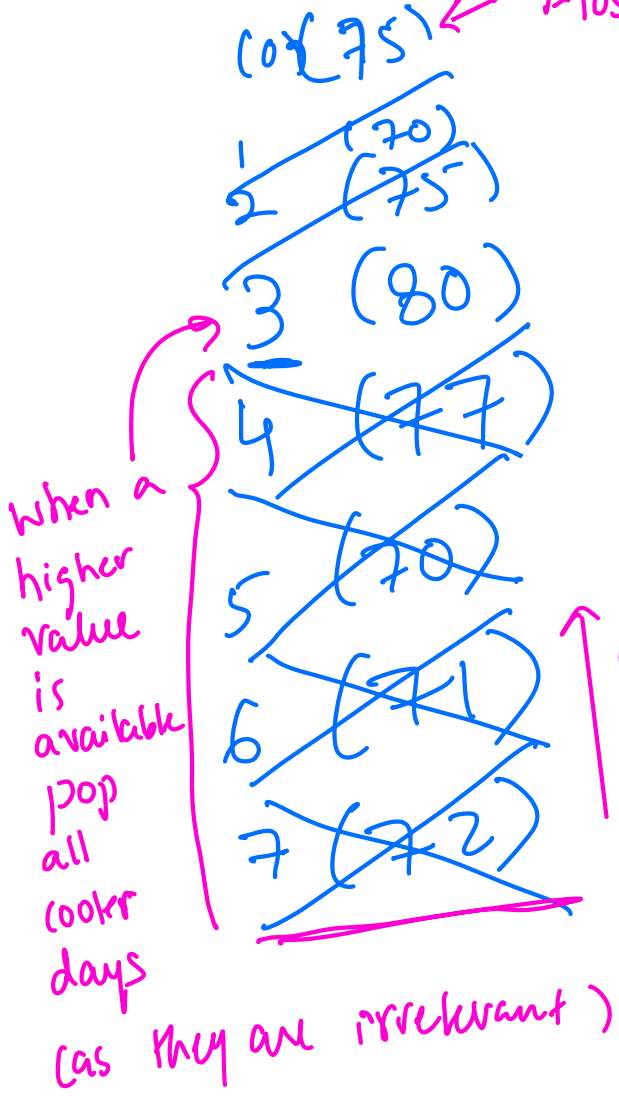
Worst case:

Temp:	80	77	75	72	70	68	65	60
Day:	0	1	2	3	4	5	6	7

Typical case:

Temp:	75	70	75	80	77	70	71	72
Day:	0	1	2	3	4	5	6	7

As you process right to left, which days are worth remembering? Which days become useless? Could a helper data structure avoid the repeated work? Write your key insight.



Ideas for optimization:

- Use a variable to store the most recent max temp (when would this not work?)
- Each day could be a potential answer for a preceding day - remember it in a stack
- When a warmer day is available (when traversing right → left) all cooler days behind it become irrelevant. Pop them from the stack.

Part 3: Stack-Based Solution

Key insight: Each day could be the answer for a preceding day: remember it in a stack

When a warmer day arrives, all cooler days behind it become irrelevant: pop them.

Trace the stack for the example below

Input: temperature on each day:

Temp	65	60	62	70	68	64	66	72
Day	0	1	2	3	4	5	6	7

Ans: Num days until a warmer day:

Ans	3	1	1	4	3	1	1	0
Day	0	1	2	3	4	5	6	7

Show the state of the stack after the temperature for each day is processed!

Stack								
Top								
Bottom								

Iteration 0 1 2 3 4 5 6 7

Note: Stack is monotonically decreasing

The visualization above shows how the stack evolves with every iteration. Provides a trace of the algorithm I coded in class.

Key insight → We never pop more than we push, one push per iteration ⇒ Total pop = O(n)

1. Complete the code to capture the stack-based solution from the previous page as the input vector is traversed right to left:

```
#include <vector>
#include <stack>
std::vector<int> dailyTemperatures(std::vector<int>& temperatures) {
        int n = temperatures.size();
        std::vector<int> answer(n, 0);
        std::stack<int> stack;
        for (int i = n - 1; i >= 0; --i) {
```

See code in lecture repo

```
        }
        return answer;
    }
```

2. What is the time and space complexity of this solution?

See code from lecture with annotated complexity analysis

$T(n) = O(n)$

$S(n) = O(n)$ (Extra space used by the stack)

Next steps:

- Attempt a different solution to this problem, traversing the input vector left to right.
- Discuss your solutions with the course staff in office hours