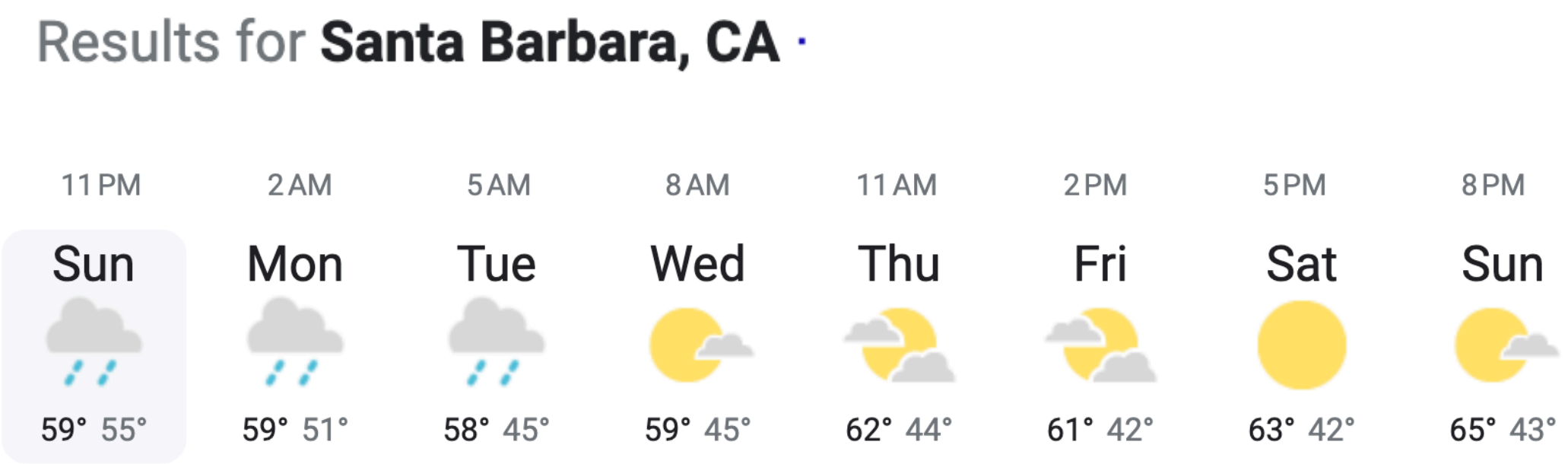
STACKS

Problem Solving with Computers-II

Announcements

- Midterm next week, May 8 (Thursday)!
  - Closed book, closed notes
  - Practice problems available in Canvas
  - All topics covered so far including this week's lectures
  - Data structures covered: Linked lists, BST, stacks and queues
  - Labs 1 - 4 and pa01
  - Leetcode problem sets 1- 3



https://leetcode.com/problems/daily-temperatures/

stack<int> s

Empty stack 

Operations: **push()** **pop()** **top()**

stack<int> s

s.push(70)

Operations: **push()** **pop()** **top()**

stack<int> s

s.push(70)

s.push(50)

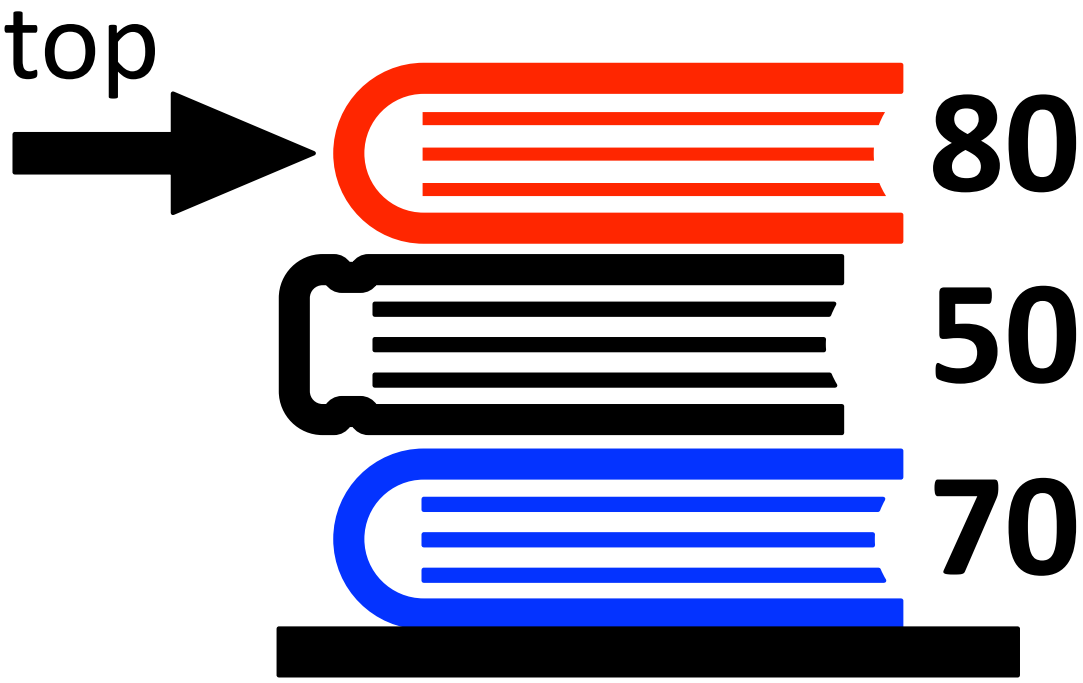
Operations: **push()** **pop()** **top()**

```
stack<int> s
s.push(70)
s.push(50)
s.push(80)
```

Operations: **push()** pop() top()

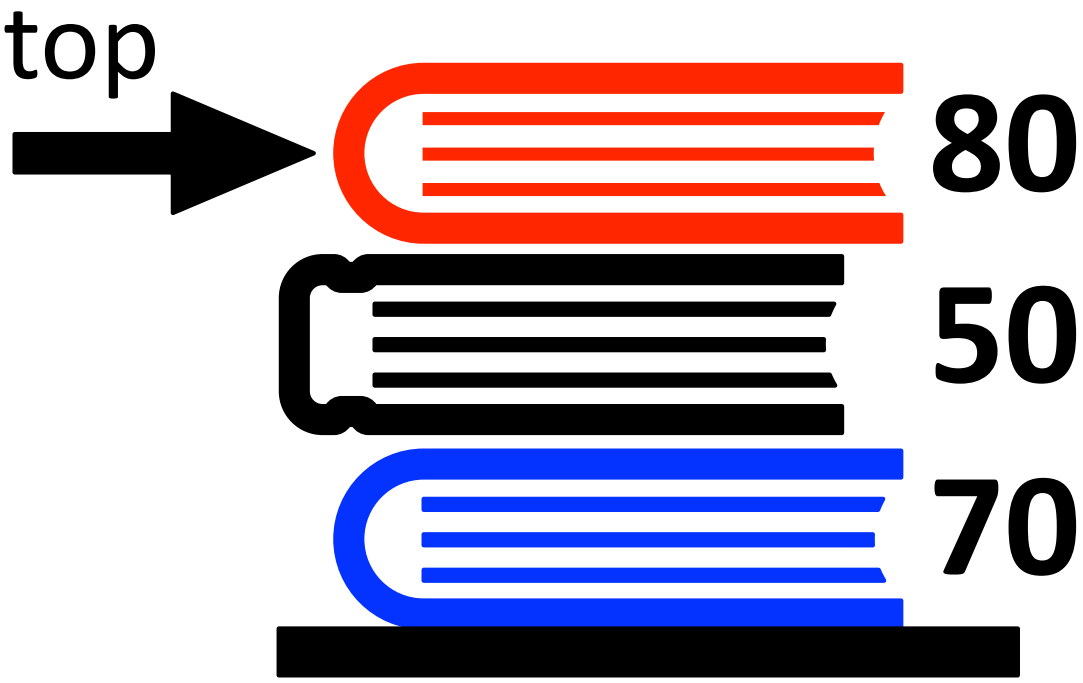
```
stack<int> s
s.push(70)
s.push(50)
s.push(80)
```

s.top() returns 80



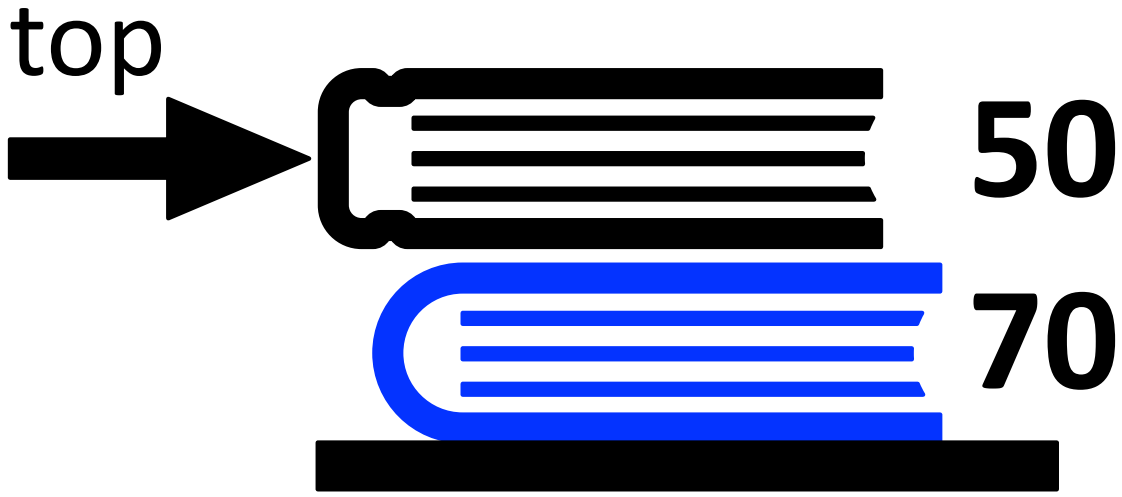
Operations: push() pop() **top()**

```
stack<int> s
s.push(70)
s.push(50)
s.push(80)


s.top()
```

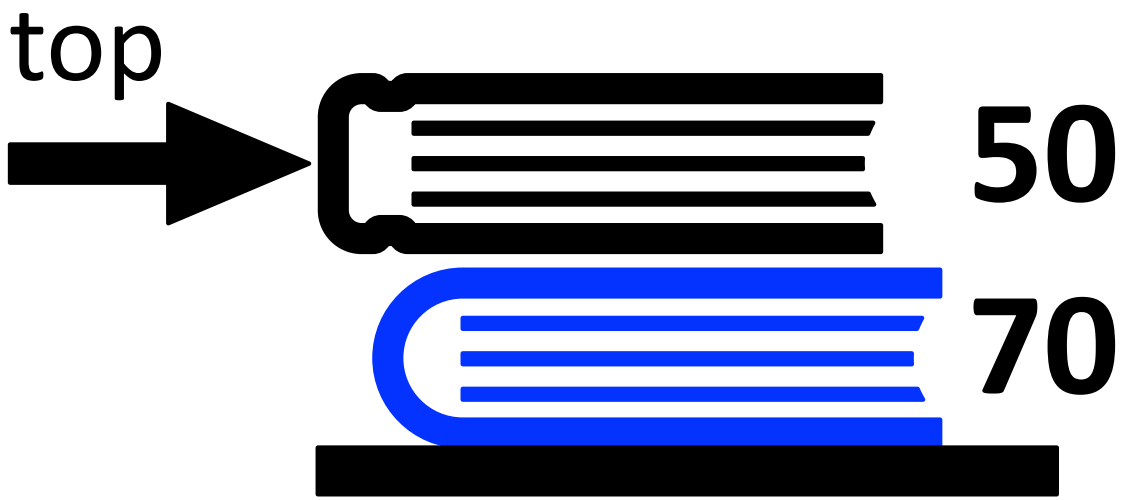


s.pop() removes value that was pushed in *last*

```
stack<int> s
s.push(70)
s.push(50)
s.push(80)


s.top()
```



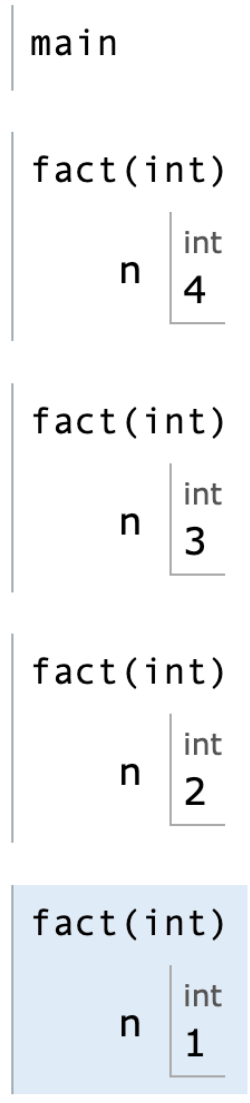
s.pop() removes value that was pushed in *last*



The Last value In is the First value Out (LIFO)

```
1  #include <iostream>
2  using namespace std;
3
4  int fact(int n){
5    if(n <= 1) return 1;
6    return n * fact(n - 1);
7  }
8
9  int main() {
10   cout<< fact(4) << endl;
11   return 0;
12 }
```

The call stack:

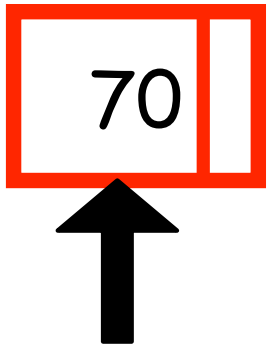


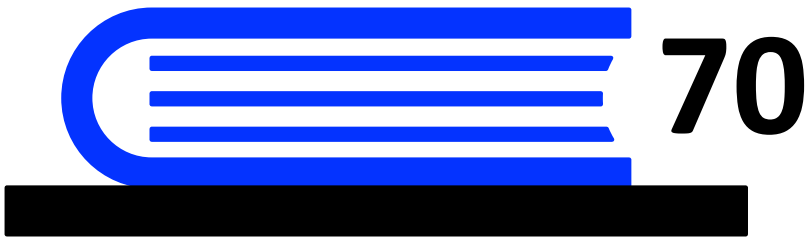
The Last value In is the First value Out (LIFO)

Implement using vector or linked list

Empty stack

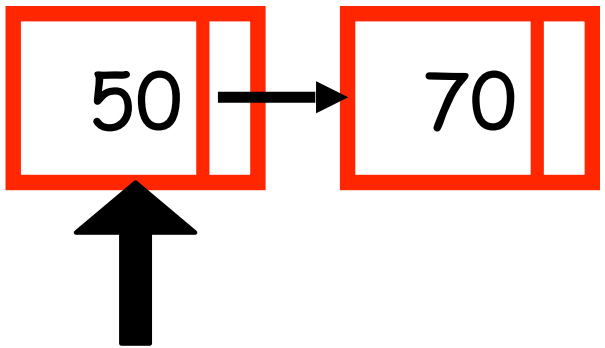
Stack Abstract Data Type





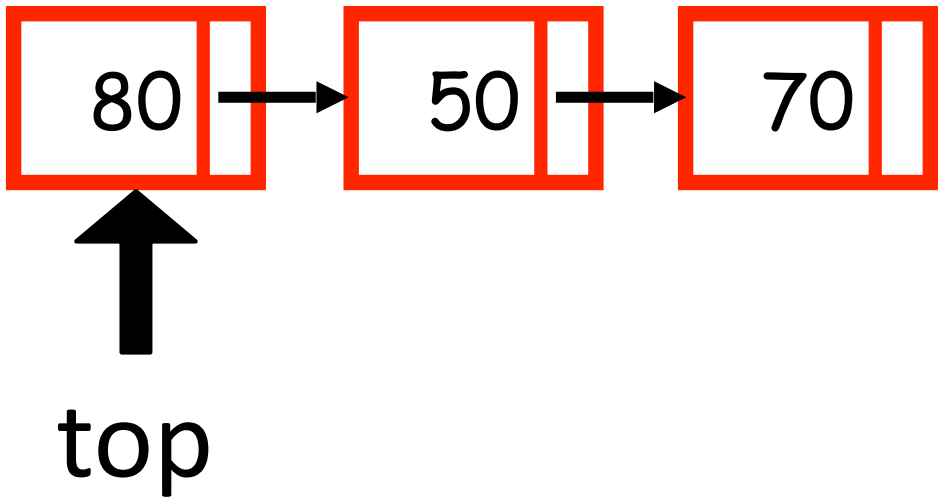
s.push(70)

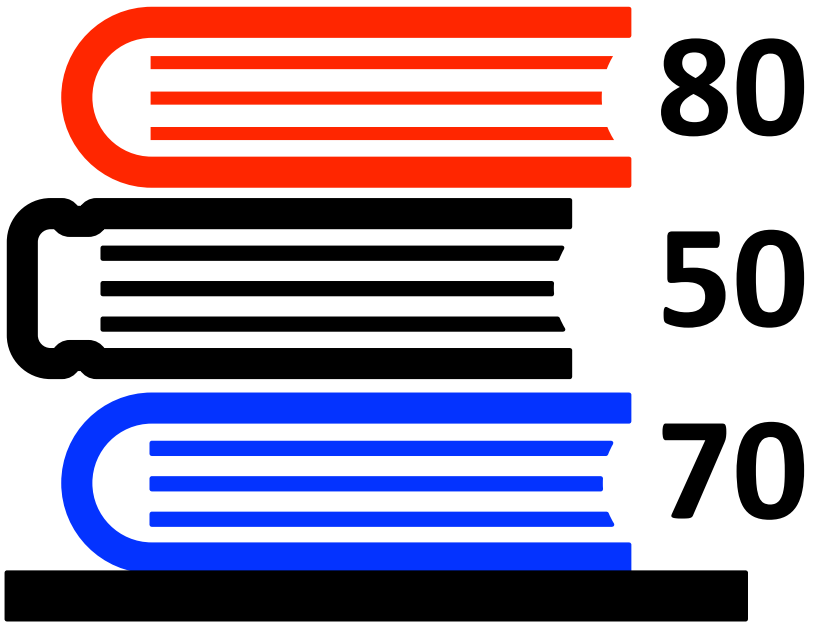
Stack Abstract Data Type



s.push(50)

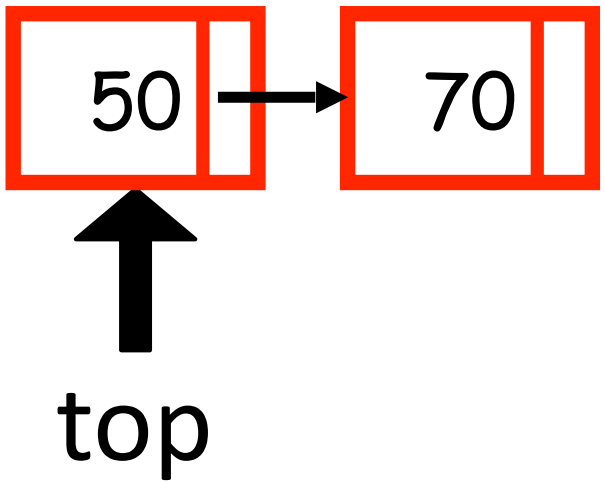
Stack Abstract Data Type





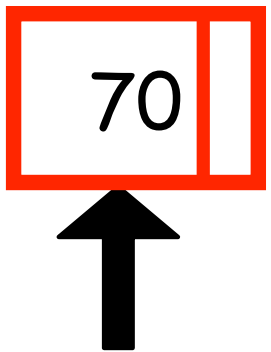
s.push(80)

Stack Abstract Data Type



s.pop()

Stack Abstract Data Type



s.pop()

Stack Abstract Data Type

Why implement a stack at all?
After all a stack is a vector or linked list with a
reduced set of operations

Stack has only three operations: **push()** **pop()** **top()**

Why implement a stack at all?
After all a stack is a vector or linked list with a **reduced set of operations**

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

Stack has only three operations: **push()** **pop()** **top()**

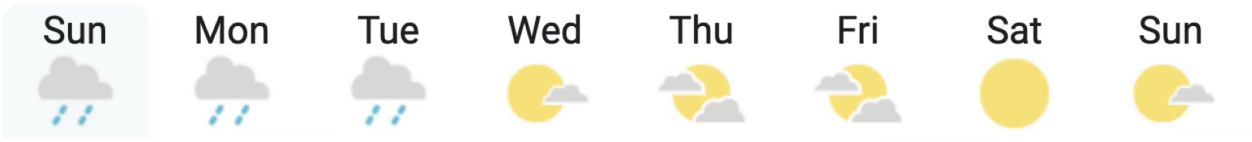
Stacks and Santa Barbara Weather Puzzle

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:



Day:	0	1	2	3	4	5	6	7
Temp:	59	59	58	59	62	61	63	65

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

Approach: Check all future days until a warmer one is found.

Ans: [4, 3, 1, 1, 2, 1, 1, 0]

0 1 2 3 4 5 6 7

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

Fill in the blanks to complete pseudocode:

```
Unset
  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

1. How many comparisons did you make to get the answer for day 0 for the given input? ____T(n) = 0 + 1 + 2 + ... + (n-1) = n(n-1)/2 ≈ O(n²) worst case
2. Write an 8-day temperature input vector that incurs the worst-case running time
   [60, 59, 58, 50, 40, 40, 40, 40]
   - Total comparisons for worst-case input:
   - Why worst?
3. Find the worst-case time and space complexity expressed in Big-O

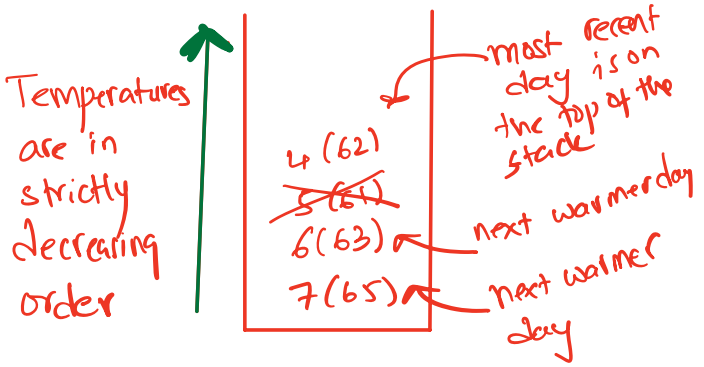
   O(n²) O(1)

To come up with an optimized solution consider the cases where the naive approach does redundant work.
e.g. To compute the answer for Day 1, we already looked at Day 2 & Day 3 temperatures & we would know that these days are not warmer than Day 1. Now for Day 0, the naive approach does not take advantage of this information, revisiting Day 2 & Day 3 values unnecessarily. So, there is room for improvement.....

High-level idea of using a monotonic stack to improve the running time

Big picture idea: Keep track of each day that can be useful to compute the answer for a preceding day in a stack. The values in the stack are indices of days whose temperature is monotonically decreasing. The most recent temp value is on the top of the stack (also most relevant)

So we make use of a monotonic stack which allows efficient queries for the next warmer day



Part 4: Stack-Based Solution

The refined solution takes O(n) for Day 0 (answer[0] = 4). Can we make it O(1) by tracking useful temperatures?

If we parse the temperatures from right to left, every day we encounter could be a potential answer (for some preceding day) — **remember potential answers in a stack!**

Complete the table to compute the answer for each day, traversing the input vector right to left and updating the stack after each iteration.

Input: temperature on each day:

Temp	59	59	58	59	62	61	63	65
Day	0	1	2	3	4	5	6	7

Output: Num days until a warmer day:

Num days	4	3	1	1	2	0	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	D0	D1	D2	D3	D4	D5	D6	D7
Top	0(59)	1(59)	2(58)	3(55)	4(62)	5(61)	6(63)	7(65)
	4(62)	4(62)	4(62)	4(62)	6(63)	6(63)	7(65)	
Bottom	6(63) 7(65)	6(63) 7(65)	6(63) 7(65)	6(63) 7(65)	7(65)	7(65)		

State of the stack after each day is processed

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; days; // Stack stores indices of days
        for (int i = n - 1; i >= 0; --i) {
            while ( ! days.empty() &&
                    temperatures[i] >= temperatures[ days.top() ]
                day.pop();
            }
            if(!days.empty()) answer[i] = days.top() - i;
            days.push(i); // Every key is pushed only
                          once. O(1) per iteration
        }
        return answer;
    }
```

Variable per iteration
Don't upper-bound per iteration

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

Time complexity:

Since each key is pushed once, the total number of pop() operations cannot exceed n. Overall cost of pop() = O(n)

The condition check for the while loop is done as many times as the number of pop() operations + one additional time per iteration of the for loop. So overall the while condition is checked at most 2n times

Overall cost of while check = O(n)

So T(n) = O(n) + O(n) + O(n)
to initialize while+pop push+answer calculation
= O(n)

Space complexity = O(n) (Extra space used for 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