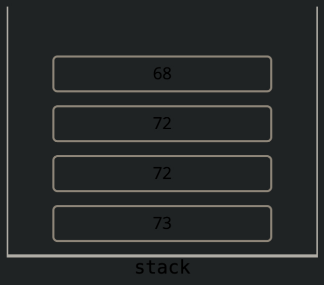
← Back to Home

Two Pointers
Sliding Window
Intervals
Stack
Overview
Valid Parentheses
Decode String
Longest Valid Parentheses
Monotonic Stack
Overview
Daily Temperatures
Largest Rectangle in Histog...
Linked List
Binary Search
Heap
Depth-First Search
Breadth-First Search
Backtracking
Graphs
Dynamic Programming
Greedy Algorithms

← Longest Valid Paren... Daily Temperatures →

Monotonic Stack

A **monotonic stack** is a special type of stack in which all elements on the stack are sorted in either descending or ascending order. It is used to solve problems that require finding the next greater or next smaller element in an array.



A montonically decreasing stack

Problem: Next Greater Element

DESCRIPTION

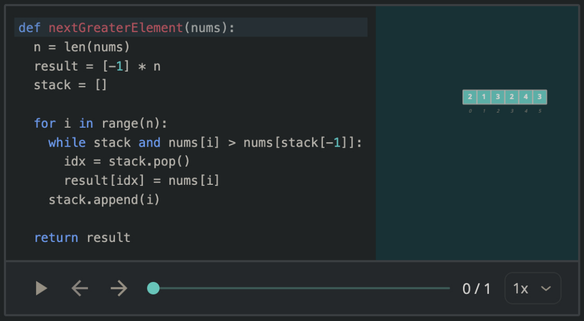
Given an array of integers, find the next greater element for each element in the array. The next greater element of an element x is the first element to the right of x that is greater than x. If there is no such element, then the next greater element is -1.

Example
Input: [2, 1, 3, 2, 4, 3]
Output: [3, 3, 4, 4, -1, -1]

The solution iterates over each index in the input array. For each index, it checks if the element at that index is the next greater element for any previous elements in the array. In order to perform that check efficiently, we'll use a **monotonic decreasing stack**.

Initialization

We start by initializing our stack and our results array, with each value in the results array initialized to -1. Our stack stores the indexes of the elements in the input array that have not yet found their next greater element.

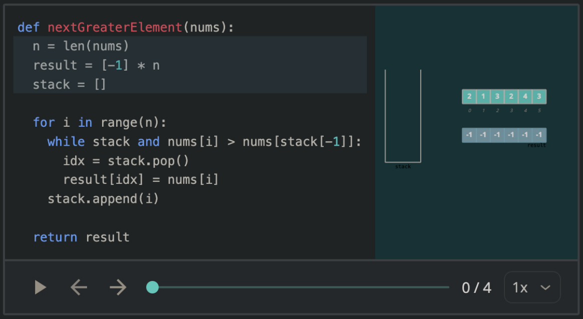


```
def nextGreaterElement(nums):
  n = len(nums)
  result = [-1] * n
  stack = []

  for i in range(n):
    while stack and nums[i] > nums[stack[-1]]:
      idx = stack.pop()
      result[idx] = nums[i]
    stack.append(i)

  return result
```

Iteration

We then iterate over the input array. To check if the current element `nums[i]` is the next greater element for any of the previous elements in the array, we compare the current element with the element at the index at the top of the stack `nums[stack[-1]]`.

If the stack is empty, or if `nums[i]` is less than `nums[stack[-1]]`, we push the current index onto the stack.



```
def nextGreaterElement(nums):
  n = len(nums)
  result = [-1] * n
  stack = []

  for i in range(n):
    while stack and nums[i] > nums[stack[-1]]:
      idx = stack.pop()
      result[idx] = nums[i]
    stack.append(i)

  return result
```

Pushing indexes 0 and 1 onto the stack

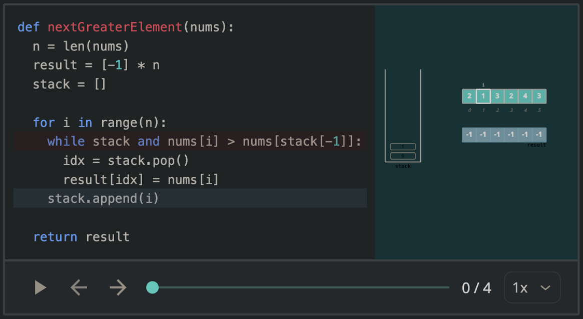
Recall that the stack contains the indexes of the elements in the input array that have not yet found their next greater element. At this point, we can see that the values at each of the indexes on the stack (i.e. `nums[0]` and `nums[1]`) are monotonically decreasing. This property allows us to check if `nums[i]` is the next greater element for any of the indexes on the stack efficiently.

If `nums[i]` is smaller than `nums[stack[-1]]`, because the stack is monotonically decreasing, **we also know that `nums[i]` is not the next greater element for any of the other indexes on the stack as well**, so we can push index `i` onto the stack.

Processing Next Greater Elements

If the `nums[i]` is greater than `nums[stack[-1]]`, then we have found the next greater element for the index `stack[-1]`. So we pop that index from the stack (`idx`), and update `results[idx]` to be `nums[i]`.

Because it is still possible for `nums[i]` to be the next greatest element for the remaining indexes on the stack, we have to repeat this processing operation until `nums[i]` is not greater than `nums[stack[-1]]`, at which point we have finished processing all the indexes for which `nums[i]` is the next greatest element, so we push `i` onto the stack.



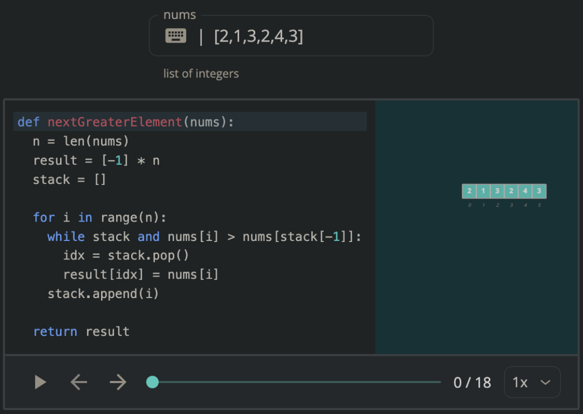
```
def nextGreaterElement(nums):
  n = len(nums)
  result = [-1] * n
  stack = []

  for i in range(n):
    while stack and nums[i] > nums[stack[-1]]:
      idx = stack.pop()
      result[idx] = nums[i]
    stack.append(i)

  return result
```

Processing indexes for which 3 is the next greatest element

> Popping all the elements that are smaller than `nums[i]` from the stack before pushing `i` ensures that the stack stays monotonically decreasing.

This process continues until the end of the input array, at which point the results array contains the next greater element for each element in the input array, or -1 if there is no such element.

Solution



nums

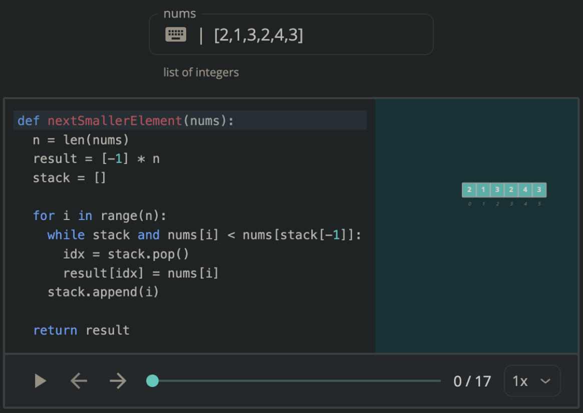
[2,1,3,2,4,3]

list of integers

```
def nextGreaterElement(nums):
  n = len(nums)
  result = [-1] * n
  stack = []

  for i in range(n):
    while stack and nums[i] > nums[stack[-1]]:
      idx = stack.pop()
      result[idx] = nums[i]
    stack.append(i)

  return result
```

Next Smaller Element

Following the same pattern, we can use a **monotonically increasing stack** to solve problems that require finding the next smaller element in an array.



nums

[2,1,3,2,4,3]

list of integers

```
def nextSmallerElement(nums):
  n = len(nums)
  result = [-1] * n
  stack = []

  for i in range(n):
    while stack and nums[i] < nums[stack[-1]]:
      idx = stack.pop()
      result[idx] = nums[i]
    stack.append(i)

  return result
```

Practice Problems

For more practice with problems that use a monotonic stack, try:

Daily Temperatures
Leetcode | Solution

Largest Rectangle in Histogram
Leetcode | Solution

Buildings with an Ocean View
Leetcode

Next: Daily Temperatures →

Add a comment...

Anonymous

Post As Tariq Williams

No comments yet.

Blog

Behavioral Interview Examples

Interviewing for Experienced SWEs

All Blog Posts

Compare Us

Compare to Interviewing.io

Links

FAQ

Schedule Mock Interviews

Pricing

Become a Coach

Our Coaches

Learn System Design

Learn DSA

Legal

Terms and Conditions

Privacy Policy

Contact

support@hellointerview.com

7511 Greenwood Ave North
Unit #4238 Seattle, WA 98103