<https://leetcode.com/problems/next-greater-element-i/description/>

The **next greater element** of some element x in an array is the **first greater** element that is **to the right** of x in the same array.

You are given two **distinct 0-indexed** integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the **next greater element** of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array* ans *of length* nums1.length *such that* ans[i] *is the* ***next greater element*** *as described above.*

**Example 1:**

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.

- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

**Example 2:**

Input: nums1 = [2,4], nums2 = [1,2,3,4]

Output: [3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.

- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the answer is -1.

**Constraints:**

1 <= nums1.length <= nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 104

All integers in nums1 and nums2 are **unique**.

All the integers of nums1 also appear in nums2.

 **Follow up:** Could you find an O(nums1.length + nums2.length) solution?

**Attempt 1: 2023-03-15**

**Solution 1: Brute Force (60 min)**

**Wrong Solution: 'j' start with 'i + 1' won't work, because even nums1 is a subset of nums2, the index relation cannot simplified as 'i + 1'**

**Test case:**

Input: nums1 = [4,1,2] , nums2 = [1,3,4,2]

Output: [-1,4,-1]

Expected: [-1,3,-1]

class Solution {

    public int[] nextGreaterElement(int[] nums1, int[] nums2) {

        int[] result = new int[nums1.length];

        Arrays.fill(result, -1);

        for(int i = 0; i < nums1.length; i++) {

            for(int j = i + 1; j < nums2.length; j++) {

                if(nums1[i] < nums2[j]) {

                    result[i] = nums2[j];

                    break;

                }

            }

        }

        return result;

    }

}

**Tricky part: Inner loop 'j' must start with "map.get(nums1[i]) + 1", 'j' start with 'i + 1' won't work**

**Q: Why we have to use map to store 'num' itself not 'num' index like L503.Next Greater Element II or L739.Daily Temperatures ?**

**A: The critical point why we can use map to store relationship between current number and its first larger number in nums2 is because statement "You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2." contains two key words "distinct" and "subset", map is the perfect data structure to handle 'distinct' relationship and also because nums1 is subset of nums2, the number of nums1 in nums2's index are undetermined yet, we have to use map to determine the number of nums1 in nums2's index first, then the next greater number index will only after that index**

class Solution {

    public int[] nextGreaterElement(int[] nums1, int[] nums2) {

        Map<Integer, Integer> map = new HashMap<Integer, Integer>();

        for(int i = 0; i < nums2.length; i++) {

            map.put(nums2[i], i);

        }

        int[] result = new int[nums1.length];

        Arrays.fill(result, -1);

        for(int i = 0; i < nums1.length; i++) {

            for(int j = map.get(nums1[i]) + 1; j < nums2.length; j++) {

                if(nums1[i] < nums2[j]) {

                    result[i] = nums2[j];

                    break;

                }

            }

        }

        return result;

    }

}

Time Complexity : O(N^2)

Space Complexity : O(1)

**Refer to**

<https://leetcode.com/problems/next-greater-element-i/solutions/97595/java-10-lines-linear-time-complexity-o-n-with-explanation/comments/200738>

我是完全没想到这种方法，只能老老实实map循环。但是它的时间复杂度应该不是O(n)，因为只是减少了内循环，没有完全取消内循环。

public int[] nextGreaterElement(int[] nums1, int[] nums2) {

    int[] re = new int[nums1.length];

    Map<Integer, Integer> m = new HashMap<>();

    for(int i = 0;i < nums2.length;i++) {

        m.put(nums2[i], i);

    }

    for(int i = 0;i < nums1.length;i++) {

        re[i] = -1;

        for(int j = m.get(nums1[i]) + 1;j < nums2.length;j++) {

            if(nums2[j] > nums1[i]) {

                re[i] = nums2[j];

                break;

            }

        }

    }

    return re;

}

**Refer to**

<https://leetcode.com/problems/next-greater-element-i/solutions/1579935/java-easy-solution-brute-and-optimal-stack-diagrammatic-explanation/>

public int[] nextGreaterElementBrute(int[] nums1, int[] nums2) {

int[] ans = new int[nums1.length];

for(int i = 0; i < nums1.length; i++) {

int greaterIdx = -1, j = nums2.length - 1;

while(j >= 0 && nums2[j] != nums1[i]) {

if(nums2[j] > nums1[i]){

greaterIdx = nums2[j];

}

j--;

}

ans[i] = greaterIdx;

}

return ans;

}

Time Complexity : O(N^2)

Space Complexity : O(1)

**Explanation**

This is actually a bit modified brute force, in this the logic is to start searching backwards until nums2[j] != nums1[i] check fails, and update greaterIdx to nums2[j] if nums2[j] > nums1[i].

Then update ans[i] to greaterIdx, it will be -1 in case no greater element and value if exists.

Then return ans

**Solution 2: Decreasing Monotonic Stack(30 min)**

**Decreasing Monotonic Stack definition: All the elements in the stack are in decreasing order from bottom to top. The determine of "decreasing monotonic stack" here is based on "in process status",  NOT the "final storage status" of stack.**

**Store number not index on Stack**

class Solution {

    public int[] nextGreaterElement(int[] nums1, int[] nums2) {

        Stack<Integer> stack = new Stack<Integer>();

        Map<Integer, Integer> map = new HashMap<Integer, Integer>();

        for(int num : nums2) {

            while(!stack.isEmpty() && stack.peek() < num) {

                map.put(stack.pop(), num);

            }

            stack.push(num);

        }

        int[] result = new int[nums1.length];

        for(int i = 0; i < nums1.length; i++) {

            result[i] = map.getOrDefault(nums1[i], -1);

        }

        return result;

    }

}

Time Complexity : O(N)

Space Complexity : O(N)

**Refer to**

<https://leetcode.com/problems/next-greater-element-i/solutions/1579935/java-easy-solution-brute-and-optimal-stack-diagrammatic-explanation/>

**Optimal**

Time - **O(n)** where n is length of nums2 as it is equal to or greater than nums1,

**Explanation**This approach uses stack, the main point is to store next greater element for all the elements in nums2 array. It does that by following steps:-

In this we traverse nums2 array and add the element to stack if either stack is empty or the element is smaller than the top element.

If case occurs when the element is greater then top in the stack then that means current element is the next greater element for that top element in stack, remove that and add it's entry in map.

Repeat this until either stack becomes empty or that element is no longer larger than top in stack.

Now iterate through nums1 and put entries in ans array by fetching it from map.

return ans

Here are some examples for better understanding

public int[] nextGreaterElement(int[] nums1, int[] nums2) {

int[] ans = new int[nums1.length];

Stack<Integer> stack = new Stack<>();

HashMap<Integer, Integer> map = new HashMap<>();

// find out all the next greater elements in nums2 array

for(int num: nums2) {

// if num is greater than top elements in stack then it is the next greater element in nums2

while(!stack.isEmpty() && num > stack.peek()) {

map.put(stack.pop(), num);

}

// then add num to stack

stack.add(num);

}

int i = 0;

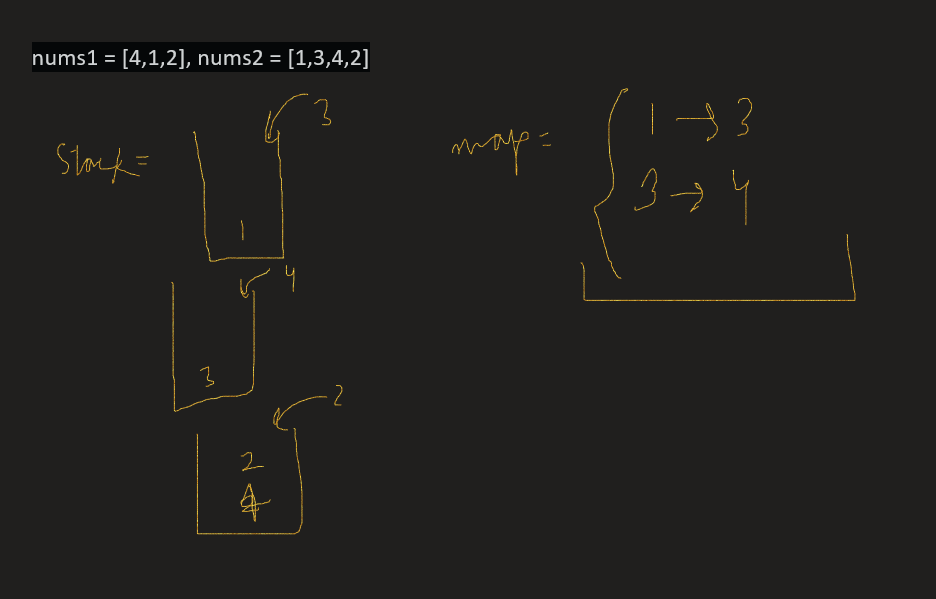
for(int num : nums1) {

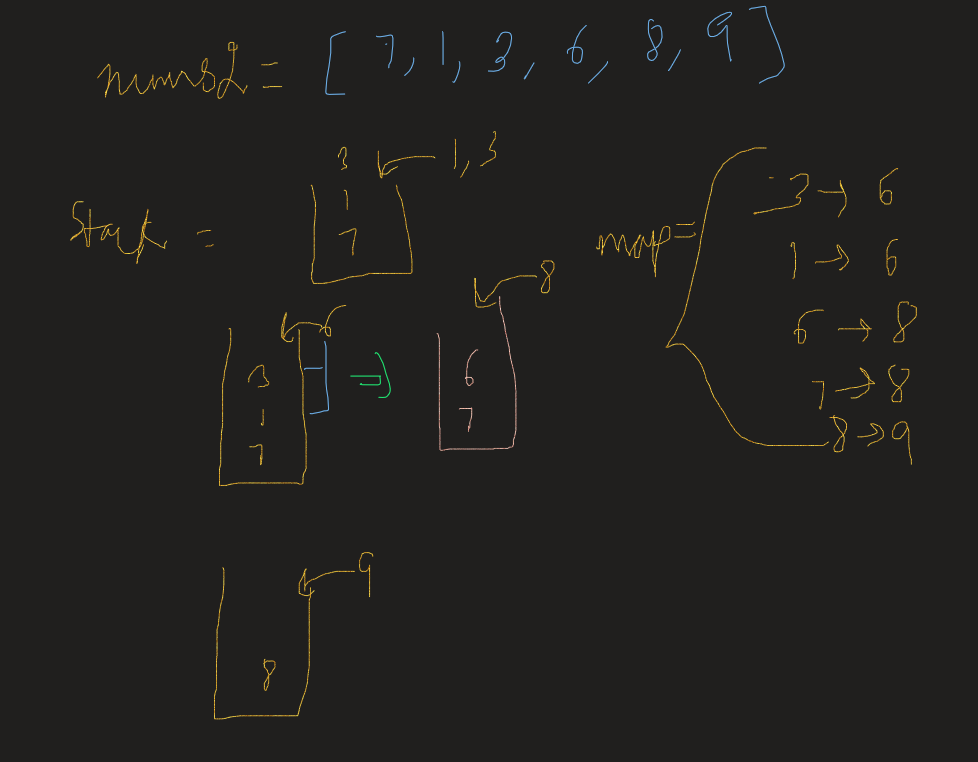
ans[i++] = map.getOrDefault(num, -1);

}

return ans;

}





**Refer to**

<https://leetcode.com/problems/next-greater-element-i/solutions/97595/java-10-lines-linear-time-complexity-o-n-with-explanation/>

Key observation: Suppose we have a decreasing sequence followed by a greater numberFor example [5, 4, 3, 2, 1, 6] then the greater number 6 is the next greater element for all previous numbers in the sequence

We use a stack to keep a **decreasing** sub-sequence, whenever we see a number x greater than stack.peek() we pop all elements less than x and for all the popped ones, their next greater element is x

For example [9, 8, 7, 3, 2, 1, 6]

The stack will first contain [9, 8, 7, 3, 2, 1] and then we see 6 which is greater than 1 so we pop 1 2 3 whose next greater element should be 6

public int[] nextGreaterElement(int[] findNums, int[] nums) {

Map<Integer, Integer> map = new HashMap<>(); // map from x to next greater element of x

Stack<Integer> stack = new Stack<>();

for (int num : nums) {

while (!stack.isEmpty() && stack.peek() < num)

map.put(stack.pop(), num);

stack.push(num);

}

for (int i = 0; i < findNums.length; i++)

findNums[i] = map.getOrDefault(findNums[i], -1);

return findNums;

}

**Solution 3: Decreasing Monotonic Stack(30 min)**

**Similar as Solution 2 but traversal from right to left**

**Store number not index on Stack**

class Solution {

    public int[] nextGreaterElement(int[] nums1, int[] nums2) {

        Stack<Integer> stack = new Stack<Integer>();

        Map<Integer, Integer> map = new HashMap<Integer, Integer>();

        for(int i = nums2.length - 1; i >= 0; i--) {

            while(!stack.isEmpty() && stack.peek() < nums2[i]) {

                stack.pop();

            }

            // If stack not empty, the peek value is the first value

            // larger than current number in nums2

            map.put(nums2[i], stack.isEmpty() ? -1 : stack.peek());

            stack.push(nums2[i]);

        }

        int[] result = new int[nums1.length];

        for(int i = 0; i < nums1.length; i++) {

            result[i] = map.get(nums1[i]);

        }

        return result;

    }

}

**Refer to**

<https://leetcode.com/problems/next-greater-element-i/solutions/97595/java-10-lines-linear-time-complexity-on-with-explanation/comments/102058>

Same idea. But I think loop from right to left is a little bit clearer.

public class Solution {

    public int[] nextGreaterElement(int[] findNums, int[] nums) {

        Map<Integer, Integer> map = new HashMap<>();

        Stack<Integer> stack = new Stack<>();

        for(int i = nums.length-1; i>=0; i--){

            while(!stack.empty() && nums[i]>stack.peek()) stack.pop();

            map.put(nums[i], (stack.empty())? -1 : stack.peek());

            stack.push(nums[i]);

        }

        for(int i = 0; i<findNums.length; i++){

            findNums[i] = map.get(findNums[i]);

        }

        return findNums;

    }

}

**Refer to**

[L739.Daily Temperatures](note://6FB3A11E2B7746DCA411301F941DD948)