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

Given an array of integers temperatures represents the daily temperatures, return *an array* answer *such that* answer[i] *is the number of days you have to wait after the* ith *day to get a warmer temperature*. If there is no future day for which this is possible, keep answer[i] == 0 instead.

**Example 1:**

Input: temperatures = [73,74,75,71,69,72,76,73]

Output: [1,1,4,2,1,1,0,0]

**Example 2:**

Input: temperatures = [30,40,50,60]

Output: [1,1,1,0]

**Example 3:**

Input: temperatures = [30,60,90]

Output: [1,1,0]

**Constraints:**

* 1 <= temperatures.length <= 105
* 30 <= temperatures[i] <= 100

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

**Solution 1: Brute Force (10 min, TLE)**

class Solution {

public int[] dailyTemperatures(int[] temperatures) {

int n = temperatures.length;

int[] result = new int[n];

for(int i = 0; i < n; i++) {

for(int j = i + 1; j < n; j++) {

if(temperatures[j] > temperatures[i]) {

result[i] = j - i;

break;

}

}

}

return result;

}

}

Time Complexity : O(N^2)

Space Complexity : O(1)

**Refer to**

<https://leetcode.com/problems/daily-temperatures/solutions/1574808/c-python-3-simple-solutions-w-explanation-examples-images-2-monotonic-stack-approaches/>

We are given an array of temperatures T and are required to return an array ans such that ans[i] is equal to distance between i and the next greater element. If no greater element than T[i] exists after ith index, ans[i] = 0

***❌ Solution - I (Brute-Force)***

Let's see how we can solve this using brute-force approach.

* For each index i, we can just iterate over the array till we either find the the 1st index j such that T[j] > T[i] or reach the end of array.
* If we find j such that T[j] > T[i], we have ans[i] = j-i.
* Otherwise, ans[i] = 0

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& T) {

vector<int> ans(size(T));

for(int i = 0; i < size(T); i++)

for(int j = i+1; j < size(T) && !ans[i]; j++) // loop till next warmer day isnt found

if(T[j] > T[i])

ans[i] = j - i;

return ans;

}

};

***Time Complexity :*** **O(N^2)**, where N is the number of elements in the input array T. In the worst case, we iterate till the end of array for each index i. So, the total number of iterations required are N-1 + N-2 + N-3 +...+ 1 = N(N-1)/2 which is equivalent to O(N2)

***Space Complexity :*** **O(1)**, ignoring the space required by the output array

**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" and the actual value based on stored 'index' on stack, NOT the "final storage status" of stack or the 'index' itself on stack.**

**To explain why saying "decreasing", for example based on given input: [73,74,75,71,69,72,76,73], during "in process status":**

**1. seen 73 [idx = 0], empty stack, push 73 [idx = 0], stack status bottom -> top = {0} represents {73}**

**2. seen 74 [idx = 1], 74 > 73, pop 73 to maintain decreasing monotonic stack, push 74 [idx = 1], stack status bottom -> top = {1} represents {74}**

**3. seen 75 [idx = 2], 75 > 74, pop 74 to maintain decreasing monotonic stack, push 75 [idx = 2], stack status bottom -> top = {2} represents {75}**

**4. seen 71 [idx = 3], 71 < 75, match decreasing monotonic stack, push 71 [idx = 3], stack status bottom -> top = {2, 3} represents {75, 71} as decreasing order**

**5. seen 69 [idx = 4], 69 < 71, match decreasing monotonic stack, push 69 [idx = 4], stack status bottom -> top = {2, 3, 4} represents {75, 71, 69} as decreasing order**

**6. seen 72 [idx = 5], 72 > 69, pop 69 to maintain decreasing monotonic stack, 72 > 71, pop 71 to maintain decreasing monotonic stack, push 72 [idx = 5], stack status bottom -> top = {2, 5} represents {75, 72} as decreasing order**

**7. seen 76 [idx = 6], 76 > 72, pop 72 to maintain decreasing monotonic stack, 76 > 75, pop 75 to maintain decreasing monotonic stack, push 76 [idx = 6], stack status bottom -> top = {6} represents {76}**

**8. seen 73 [idx = 7], 73 < 76, match decreasing monotonic stack, push 73 [idx = 7], stack status bottom -> top = {6, 7} represents {76, 73} as decreasing order**

**Store index not num itself**

class Solution {

public int[] dailyTemperatures(int[] temperatures) {

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

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

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

// Strictly '<' required

// Test case

// e.g [89,62,70,58,47,47,46,76,100,70]

// Output [8,1,5,4,1,2,1,1,0,0]

// Expected [8,1,5,4,3,2,1,1,0,0]

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

int index = stack.pop();

result[index] = i - index;

}

stack.push(i);

}

return result;

}

}

Time Complexity : O(N)

Space Complexity : O(N)

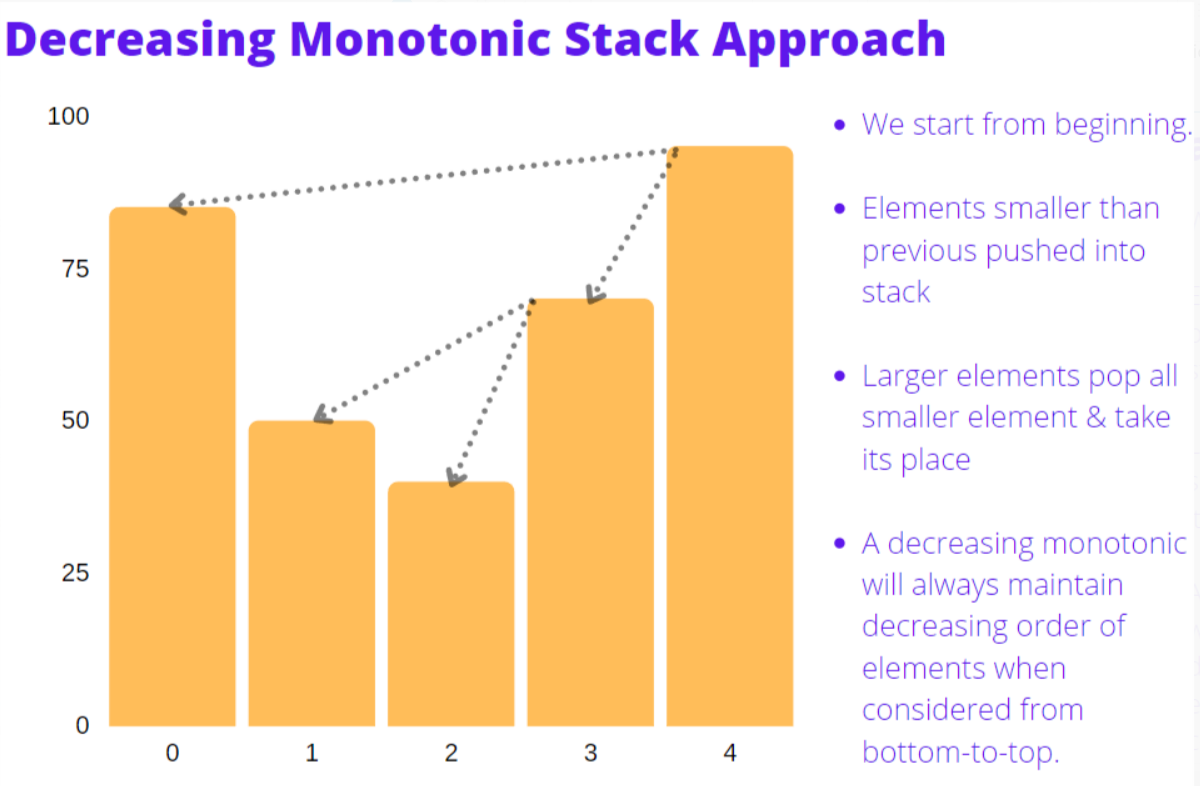
**Refer to**

<https://leetcode.com/problems/daily-temperatures/solutions/1574808/c-python-3-simple-solutions-w-explanation-examples-images-2-monotonic-stack-approaches/>

✔️ ***Solution - II (Decreasing Monotonic Stack)***

In the above solution, we can see that in the worst case, we are repeatedly iterating till the end of array either to find the next greater element at the very end or not finding it at all. This is redundant. We can optimize the above approach by observing that multiple elements can share a common warmer day. For e.g. Consider [4,3,2,1,5]. In the brute-force, we would have iterated till 5th element in every case and assigned ans[i] as difference between the indices. However, we see that all elements share 5 as the next warmer day.

Thus, the pattern we see is that we **iterate forward till we find a warmer day** and **that day will be the answer for all elements before it that are cooler** (and not found a warmer day). Thus, we can **maintain a stack consisting of indices of days which haven't found a warmer day**. The temperature at these indices will be in decreasing order since we are only storing days that haven't found a warmer next day and hence it is known as decreasing monotonic stack. Whenever we find a current day cur which is warmer, we check elements of stack and assign them the difference of indices for all those elements which have temperature of corresponding day less than T[cur].



Thus, the algorithm can be summarized as -

1. Initialize an empty stack s and ans array of size N all initialized to 0s.
2. Iterate over T from the start
3. For each current day cur, check if today's temperature T[cur] is warmer than values corresponding to previous days' indices stored in stack (T[[s.top](http://s.top)()]). Assign answer for all elements of stack for whom current day's temperature is warmer (T[cur] > T[[s.top](http://s.top)()]) and pop them off the stack.
4. Push cur onto the stack denoting that we need to find warmer next day for cur.
5. All the elements present in the stack at end don't have a next greater element. We don't have to worry about them since ans is already initialized to 0. Thus, we can directly return ans.

Consider T = [73,74,75,71,69,72,76,73]

Each row's value in below table is the final value after completion of that step.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i/ cur** | **T[i]** | **stack** | **ans** | **Description** |
| 0 | 73 | [0] | [0,0,0,0,0,0,0,0] | Stack was empty. So, we found no element for which T[i]could be assigned as next warmer day. So, just pushed current day 0to stack |
| 1 | 74 | [1] | [1,0,0,0,0,0,0,0] | Today's temperature: 74 is warmer than stack top's corresponding temp: 73. So, today can be assigned as next warmer day to stack's top(ans[0] = 1-0 = 1) and then it's popped. Lastly push today's index to stack |
| 2 | 75 | [2] | [1,1,0,0,0,0,0,0] | Again, today's temperature: 75 is warmer than stack top's corresponding temp :74. So, today can be assigned as next warmer day to stack's top(ans[1] = 2-1 = 1) and then it's popped. Push cur=2to stack |
| 3 | 71 | [2,3] | [1,1,0,0,0,0,0,0] | Today's temperature: 71 is NOT warmer than stack top's corresponding temp :75. So, just push cur=3to stack |
| 4 | 69 | [2,3,4] | [1,1,0,0,0,0,0,0] | Again, today's temperature: 69 is NOT warmer than stack top's corresponding temp :71. So just push cur=4to stack |
| 5 | 72 | [2,5] | [1,1,0,2,1,0,0,0] | Today's temperature: 72 is warmer than stack top's corresponding temp :69. Assign cur as next warmer day to stack's top(ans[4] = 5-4 = 1) & pop it.  Again, after popping, 72 is warmer than stack top's corresponding temp :71. Assign cur as next warmer day to stack's top(ans[3] = 5-3 = 2) & pop it. Lastly, push cur=5to stack |
| 6 | 76 | [6] | [1,1,4,2,1,1,0,0] | Today's temperature: 76 is warmer than stack top's corresponding temp :72. Assign ans[5] = 6-5 = 1and pop it.  Again, after popping, 72 is warmer than stack top's corresponding temp :75. Assign ans[2] = 6-2 = 4& pop it. Push cur=6to stack |
| 7 | 73 | [6,7] | [1,1,4,2,1,1,0,0] | Today's temperature: 73 is NOT warmer than stack top's corresponding temp :76. Just push cur=6to stack.  We stop here since reached the end of array |

**Code**

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& T) {

stack<int> s;

vector<int> ans(size(T));

for(int cur = 0; cur < size(T); cur++) {

while(size(s) and T[cur] > T[s.top()]) { // pop till current temp < stack's top's temp. This maintains monotonic stack

ans[s.top()] = cur - s.top(); // cur day will be next warmer day for each element that's popped

s.pop();

}

s.push(cur); // push onto stack to find next warmer day for cur later on

}

return ans;

}

};

***Time Complexity :*** **O(N)**, In the worst case, we require O(2\*N) ~ O(N) iterations.

***Space Complexity :*** **O(N)**, In the worst case, we may have decreasing elements in T and stack will store all N indices in it

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

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

**To explain why also saying "decreasing" when traversal from right to left, for example based on given input: [73,74,75,71,69,72,76,73], during "in process status":**

**1. seen 73 [idx = 7], empty stack, push 73 [idx = 7], stack status bottom -> top = {7} represents {73}**

**2. seen 76 [idx = 6], 76 > 73, pop 73 to maintain decreasing monotonic stack, push 76 [idx = 6], stack status bottom -> top = {6} represents {76}**

**3. seen 72 [idx = 5], 72 < 76, match decreasing monotonic stack, push 72 [idx = 5], stack status bottom -> top = {6, 5} represents {76, 72} as decreasing order**

**4. seen 69 [idx = 4], 69 < 72, match decreasing monotonic stack, push 69 [idx = 4], stack status bottom -> top = {6, 5, 4} represents {76, 72, 69} as decreasing order**

**5. seen 71 [idx = 3], 71 > 69, pop 69 to maintain decreasing monotonic stack, push 71 [idx = 3], stack status bottom -> top = {6, 5, 3} represents {76, 72, 71} as decreasing order**

**6. seen 75 [idx = 2], 75 > 71, pop 71 to maintain decreasing monotonic stack, 75 > 72, pop 72 to maintain decreasing monotonic stack, push 75 [idx = 2], stack status bottom -> top = {6, 2} represents {76, 75} as decreasing order**

**7. seen 74 [idx = 1], 74 < 75, match decreasing monotonic stack, push 74 [idx = 1], stack status bottom -> top = {6, 2, 1} represents {76, 75, 74} as decreasing order**

**8. seen 73 [idx = 0], 73 < 74, match decreasing monotonic stack, push 73 [idx = 0], stack status bottom -> top = {6, 2, 1, 0} represents {76, 75, 74, 73} as decreasing order**

**Store index not num itself**

class Solution {

public int[] dailyTemperatures(int[] temperatures) {

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

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

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

// Pop till current temp > stack top's temp. All popped element can never be next warmer day for any other elements

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

stack.pop();

}

// No warmer element exists if stack empty. Otherwise, assign distance between stack's top and cur day

result[i] = stack.isEmpty() ? 0 : stack.peek() - i;

// Push onto stack as it can potentially be next warmer day for rest elements

stack.push(i);

}

return result;

}

}

Time Complexity : O(N)

Space Complexity : O(N)

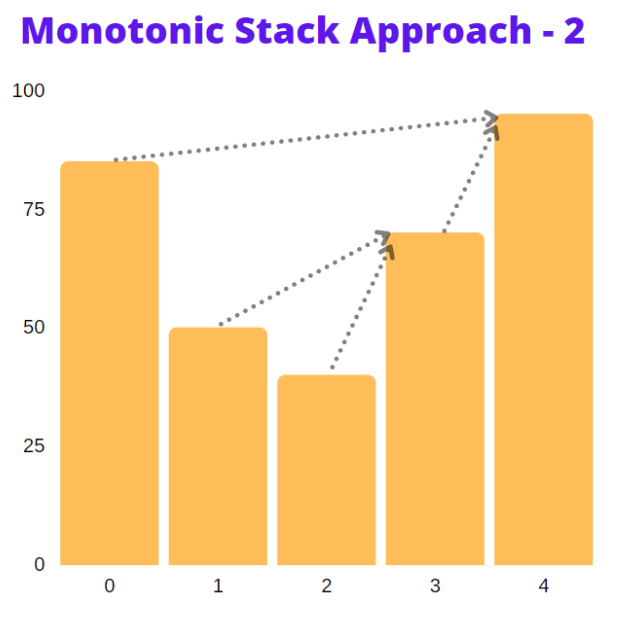
**Refer to**

<https://leetcode.com/problems/daily-temperatures/solutions/1574808/c-python-3-simple-solutions-w-explanation-examples-images-2-monotonic-stack-approaches/>

***✔️ Solution - III (Decreasing Monotonic Stack - 2)***

Another way of modelling the problem in terms of monotonic stack that some may find more intuitive is by starting from the end of the array. We again maintain a monotonic stack in this case as well which is similar to above approach, the only change is just that we start from the end.

This will again allow us to find the next warmer element for each index just by looking through the stack. Since we are maintaining a sorted stack (increasing from top-to-bottom), we know that the first element that we find in stack for curth day such that T[[s.top](http://s.top)()] > T[cur], will be next warmer day required for that element.



In the above image, we start from the end and each time assign next warmer day to be top of stack element. In the 1st approach, we instead started from beginning and only assigned next warmer day/s at once once we find a warmer day than all preceding elements. Both approach should work just fine.

The algorithm can be summarized as -

Initialize an empty stack s and ans array of size N all initialized to 0s.

Iterate over T from the end.

For each current day cur, pop values corresponding from stack that have temperature(T[[s.top](http://s.top)()]) lower than today's temperature T[cur], i.e, T[[s.top](http://s.top)()] <= T[cur]. This popping is done because these elements are cooler than T[cur] and occur later on than cur. Thus, they will never be a potential answer for elements on the left.

Now that all elements lower than T[cur] have been popped off, stack s is either empty or has some element warmer than cur.

If stack is empty, assign ans[cur] = 0 because no next warmer element exists for cur.

Otherwise, assign ans[cur] = [s.top](http://s.top)()-cur, the difference between indices of next warmer element and cur.

Then, Push cur onto the stack since it has potential to be next closest warmer day for remaining elements on left of T.

Finally, return ans.

Again, consider T = [73,74,75,71,69,72,76,73] and we get the following steps listed in table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i/ cur** | **T[i]** | **stack** | **ans** | **Description** |
| 7 | 73 | [7] | [0,0,0,0,0,0,0,0] | Stack was empty. So, we found no next warmer element for T[i]. Assign ans[7] = 0and push current day 7to stack |
| 6 | 76 | [6] | [0,0,0,0,0,0,0,0] | Stack's top's temp: 73 is less than today's temperature:76. So, pop it since it can't ever be next warmer element for any day. Again, stack is empty and so there's not next warmer element for today. Assign ans[6] = 0and push current day 6to stack |
| 5 | 72 | [6,5] | [0,0,0,0,0,1,0,0] | Stack's top's temp: 76 is warmer than today's temperature:72. So assign ans[5] = 6-5 = 1. Then, push current day 5to stack |
| 4 | 69 | [6,5,4] | [0,0,0,0,1,1,0,0] | Stack's top's temp: 72 is warmer than today's temperature:69. So assign ans[4] = 6-5 = 1.and push current day 4to stack |
| 3 | 71 | [6,5,3] | [0,0,0,2,1,1,0,0] | Stack's top's temp: 69 is less than today's temperature:71. So pop it. Now, stack's top has greater temp so we break out of loop and assign ans[3] = 5-3 = 2.and push current day 3to stack |
| 2 | 75 | [6,2] | [0,0,4,2,1,1,0,0] | Stack's top's temp: 71 is less than today's temperature:75. So pop it. Again, stack's top temp: 72is less than 75. So, pop it again. Now, stack's top temp is greater so we break out of loop and assign ans[2] = 6-2 = 4.and push current day 2to stack |
| 1 | 74 | [6,2,1] | [0,1,4,2,1,1,0,0] | Stack's top's temp: 75 is warmer than today's temperature:74. So assign ans[1] = 2-1 = 1.and push current day 1to stack |
| 0 | 73 | [6,2,1,0] | [1,1,4,2,1,1,0,0] | Stack's top's temp: 74 is warmer than today's temperature:73. So assign ans[0] = 1-0 = 1.and push current day 0to stack |

**Code**

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& T) {

stack<int> s;

vector<int> ans(size(T));

for(int cur = size(T)-1; cur >= 0; cur--) {

// pop till current temp > stack top's temp. All popped element can never be next warmer day for any other elements

while(size(s) and T[s.top()] <= T[cur]) s.pop();

ans[cur] = s.empty() ? 0 : s.top() - cur; // no warmer element exists if stack empty. Otherwise, assign distance between stack's top and cur day

s.push(cur); // push onto stack as it can potentially be next warmer day for rest elements

}

return ans;

}

};

***Time Complexity :*** **O(N)**, same as above

***Space Complexity :*** **O(N)**, In worst case, we may have increasing elements in T and stack will store all N indices in it.