Finding Next Greater Element (NGE)

Mr. Akash Yadav

Assistant Professor

Artificial Intelligence & Data Science

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Finding Next Greater Element (NGE)**

Lesson Objectives
To understand the definition and importance of the Next Greater Element problem in programming and algorithm design.
To implement both brute force and efficient stack-based code to solve for the Next Greater Element in a given array.
To analyse and compare the time and space complexity of brute force and stack-based approaches.
To identify suitable scenarios in computer science or real-life where NGE solutions are applicable and efficient.

Problem Statement:

Write a program for finding NGE NEXT GREATER ELEMENT from an array.

Introduction to the Problem

- The Next Greater Element (NGE) problem is a classic array and stack problem in computer science.
- The Next Greater Element (NGE) for an element in an array is the first element to its right that is strictly greater than itself.
- If no such element exists, its NGE is -1

❖ Example:

Input: [4, 5, 2, 25]

Output: [5, 25, 25, -1]

Concept and Background

- The Next Greater Element problem is important in data structure learning.
- It is used in stock span problems, temperature predictions, and interval analysis.

❖ **There are mainly two ways to solve this problem:**

- Naive approach or brute force method
- Stack-based approach

Algorithm/Logic

Step 1: Initialize an empty stack. Use it to keep track of potential "next greater" candidates.

Step 2: Traverse the array from right to left.

Step 3: While stack is not empty and top ≤ current element, pop the stack.

Step 4: If stack is empty, NGE = -1; else NGE = top of stack.

Step 5: Push the current element into the stack.

Step 6: Continue until all elements are processed.

Visualization

- ❑ Example:

  Let’s try to find NGE for all the elements present in the given array : arr = [6][8][0][1][3]

- ❑ Step-by-Step Stack Approach

  We process array elements from right to left and maintain a stack where the potential next greater elements are stored.

- ❑ Initialization

  Stack: []
  Result: [_, _, _, _, _]

Visualization

- **(A)** i = 4, value = 3
  1. Stack is empty. So, NGE for 3 is -1.
  2. Push 3 to stack.
  3. Result: [_, _, _, _, -1]
  4. Stack: [3]

- **(B)** i = 3, value = 1
  1. Stack's top (3) > 1, so NGE for 1 is 3.
  2. Push 1 to stack.
  3. Result: [_, _, _, 3, -1]
  1. Stack: [3][1]

Visualization

□ **(C)** i = 2, value = 0

1. Stack's top (1) > 0, so NGE for 0 is 1.
2. Push 0 to stack.
3. Result: [_, _, 1, 3, -1]
4. Stack: [3][1][0]

□ **(D)** i = 1, value = 8

1. Stack's top (0) ≤ 8; pop 0.
2. Next top (1) ≤ 8; pop 1.
3. Next top (3) ≤ 8; pop 3.
4. Now, stack is empty: NGE for 8 is -1.
5. Push 8.
6. Result: [_, -1, 1, 3, -1]
7. Stack: [8]

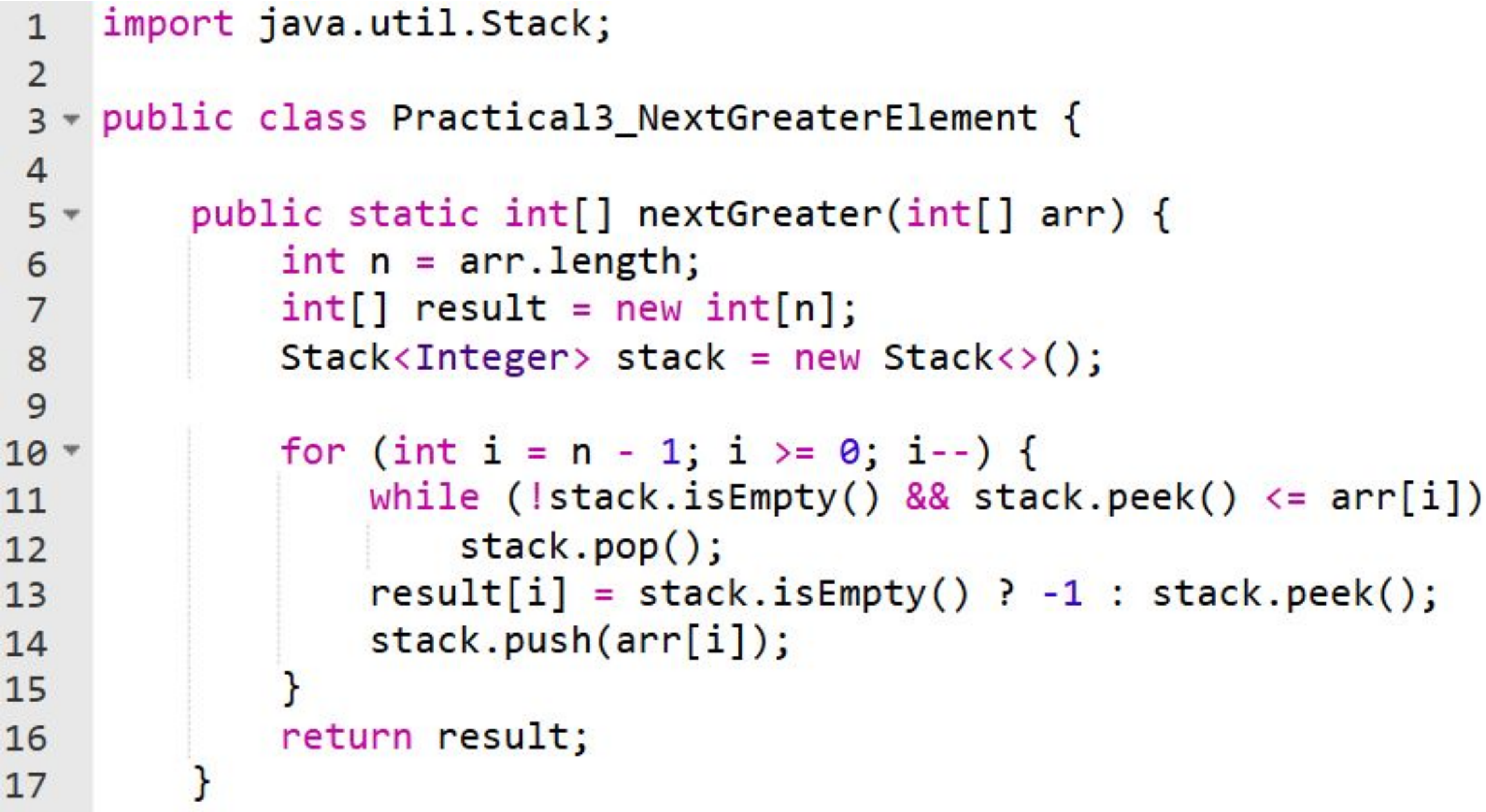
Visualization

□ **(E)** i = 0, value = 6

1. Stack's top (8) > 6, so NGE for 6 is 8.
2. Push 6.
3. Result: [8, -1, 1, 3, -1]
4. Stack: [8][6]

✔ Final Answer: [8, -1, 1, 3, -1]

Code Implementation



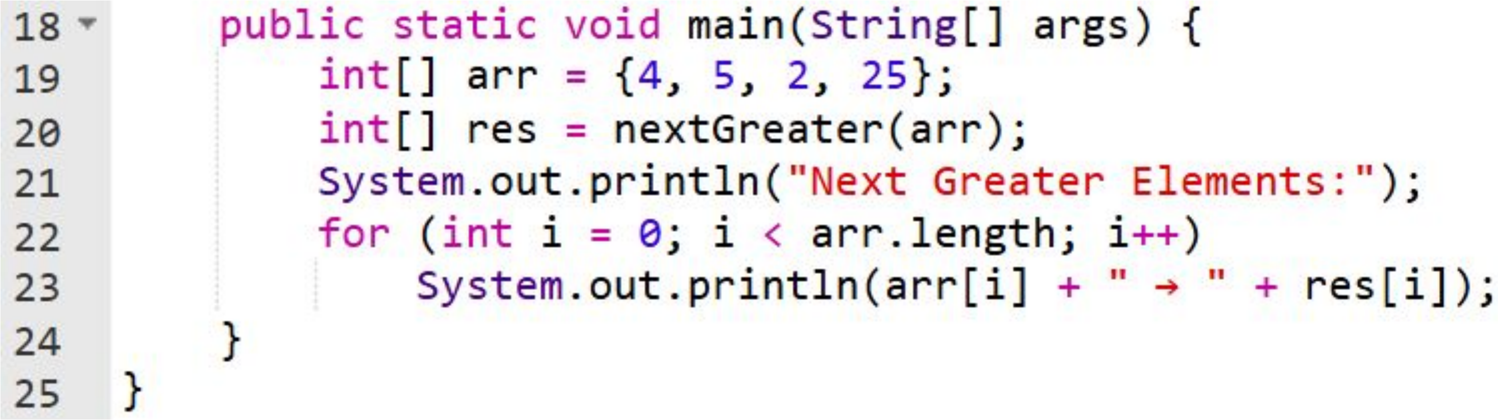
```java
import java.util.Stack;

public class Practical3_NextGreaterElement {

    public static int[] nextGreater(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i])
                stack.pop();
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }
        return result;
    }
```

Code Implementation



```java
    public static void main(String[] args) {
        int[] arr = {4, 5, 2, 25};
        int[] res = nextGreater(arr);
        System.out.println("Next Greater Elements:");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i] + " → " + res[i]);
    }
}
```

Output :

```
Next Greater Elements:

4 → 5

5 → 25

2 → 25

25 → -1
```

Time & Space Complexity

Method	Approach	Time Complexity	Space Complexity	Example Output
Brute Force	Nested Loops		O(n)	[8, -1, 1, 3, -1]
Stack	Monotonic Stack	O(n)	O(n)	[8, -1, 1, 3, -1]

Summary

- Implemented a program to find the Next Greater Element for each element in an array.
- Used a stack to efficiently track elements and determine the next greater value on the right side.
- Improved efficiency from the brute-force $O(n^2)$ approach to an optimized $O(n)$ solution.
- Demonstrated the use of stack operations in solving array-based problems where relative ordering matters.
- Reinforced understanding of LIFO behaviour and real-time element comparison in algorithmic problems.

Practice Questions:

- **1.Next Greater Element II (Problem 503):**

- 🔗 https://leetcode.com/problems/next-greater-element-ii/

Concept: The "Next Greater Element II" problem extends the concept of finding the next greater element in a linear array to a circular array.

Why Practice: Same as our class example — perfect for reinforcement.

Practice Questions:

- **2.Next Greater Element III (Problem 556):**
- 🔗 https://leetcode.com/problems/next-greater-element-iii/

Concept: The "Next Greater Element III" problem asks for the smallest integer greater than a given integer n that uses exactly the same digits as n, while fitting within a 32-bit integer; if no such number exists, return -1.

Why Practice: Strengthens your understanding of NGE problems logic.

Thanks