

code

```
public static void nextGreaterOnLeft(int[] arr) {
    int[] ans = new int[arr.length];

    Stack<Integer> st = new Stack<Integer>();
    for (int i = 0; i < arr.length; i++) {
        while ( !st.isEmpty() && st.peek() <= arr[i] ) {
            st.pop();
        }
        if (!st.isEmpty()) {
            ans[i] = st.peek();
        } else {
            ans[i] = -1;
        }
        st.push( arr[i] );
    }

    for (int i : ans) {
        System.out.print(i + " ");
    }
}
```

ensuring
all unnecessary
elements
will be
pop out

top
element

curr
element

| | | | | | | | |
|---|---|---|---|----|----|---|----|
| 7 | 3 | 7 | 7 | -1 | -1 | 2 | -1 |
|---|---|---|---|----|----|---|----|

arr

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 3 | 2 | 7 | 6 | 0 | 2 |
|---|---|---|---|---|---|---|---|

stack

| |
|---|
| 2 |
| 0 |
| 6 |
| 7 |
| 2 |
| 3 |
| 1 |
| 5 |

$5 \leq 1 \times$

$1 \leq 3 \checkmark$

$5 \leq 3 \times$

$3 \leq 2 \times$

$2 \leq 7 \checkmark$

$3 \leq 7 \checkmark$

$5 \leq 7 \checkmark$

$7 \leq 6 \times$

$6 \leq 0 \times$

$0 \leq 2 \checkmark$

ans

| | | | | | | | |
|----|---|---|---|----|---|---|---|
| -1 | 5 | 5 | 3 | -1 | 7 | 6 | 6 |
|----|---|---|---|----|---|---|---|

↳ Greater element on left (done)

↳ Greater element on right (reverse the for loop)

↳ Smaller element on left (condition is while loop will get reverse)

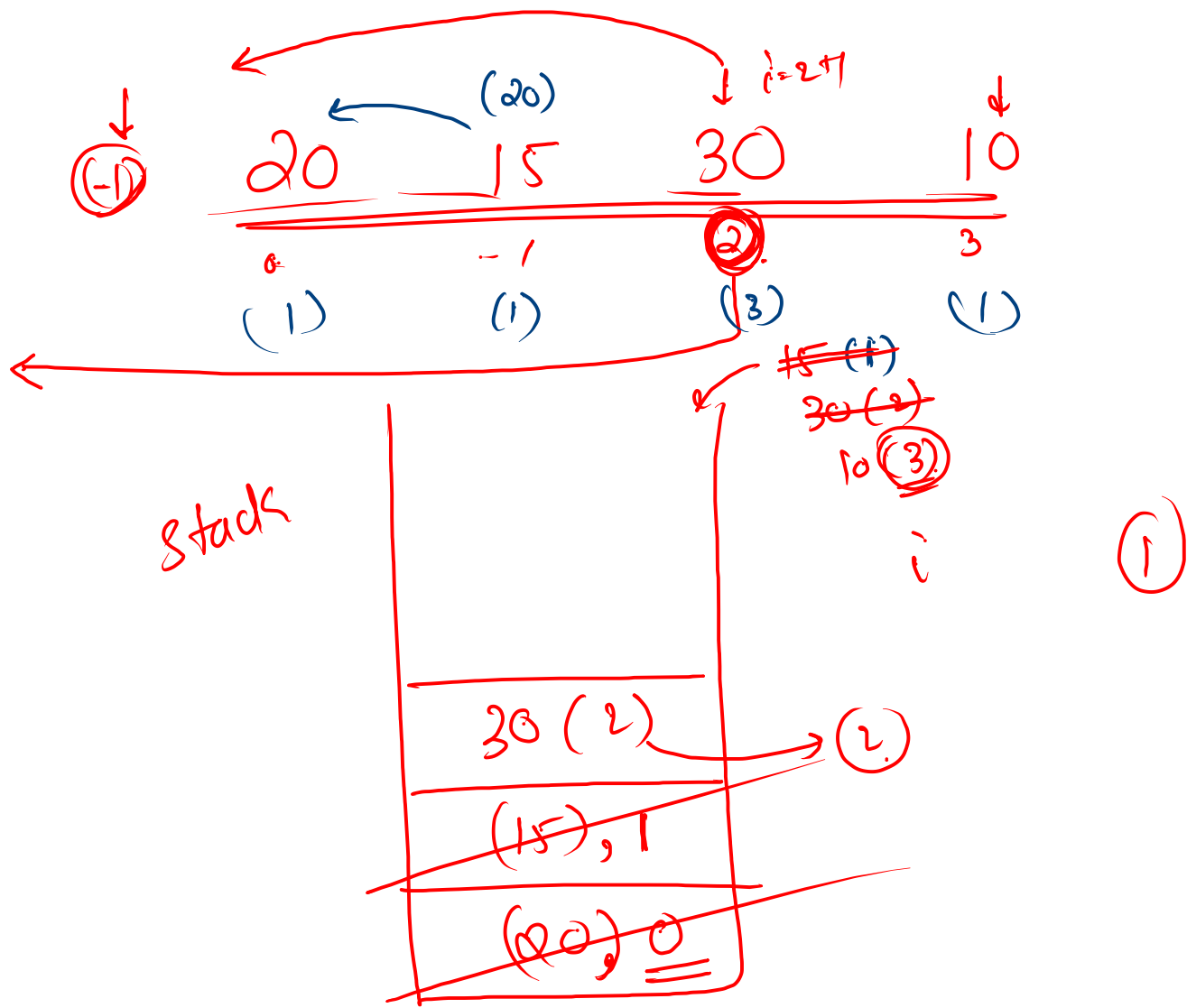
↳ Smaller element on right
↳ reverse the loop & ^{well} and the condⁿ is while as

Online Stock Spanner

| | | | | | | | |
|------|-------|------|------|------|------|-------|--|
| 7 | | | | | | | |
| 100 | 80 | 60 | 70 | 60 | 75 | 85 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | |
| 1 | 1 | 1 | 2 | 1 | 4 | 6 | |
| (-1) | (100) | (80) | (80) | (70) | (80) | (100) | |

Smart work

↳ Instead of storing the element, I will store the index of each element.



```
public static void nextGreaterOnLeft(int[] arr) {  
    int[] ans = new int[arr.length];  
  
    Stack<Integer> st = new Stack<Integer>();  
    for (int i = 0; i < arr.length; i++) {  
        while ( !st.isEmpty() && arr[st.peek()] <= arr[i] ) {  
            st.pop();  
        }  
        if (!st.isEmpty()) {  
            ans[i] = i - st.peek();  
        } else {  
            ans[i] = i + 1;  
        }  
        st.push( i );  
    }  
  
    for (int i : ans) {  
        System.out.print(i + " ");  
    }  
}
```

Handwritten annotations in red:

 - A bracket on the left side of the for loop, spanning from the loop header to the closing brace.
 - Under the expression `arr[st.peek()]` in the while loop condition, with the word "top" written below it.
 - Under the expression `arr[i]` in the while loop condition, with the word "curr" written below it.
 - Under the expression `i - st.peek()` in the if block.
 - Under the expression `i + 1` in the else block, with a checkmark to its right.
 - Under the expression `st.push(i)`.

ans

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 5 | 1 | 1 |
|---|---|---|---|---|---|---|

 ∞
 (-1)

| | | | | | | |
|----|----|----|----|-----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 30 | 40 | 20 | 25 | 100 | 80 | 60 |

\uparrow \uparrow \uparrow \uparrow \uparrow \uparrow \uparrow
 i j i j i j i

| |
|------------------|
| |
| |
| 60(6) |
| 80(5) |
| 100(4) |
| 25(3) |
| (20)2 |
| (40)1 |
| (30)0 |

(val) idx

~~30(0)~~~~40(1)~~~~20(2)~~~~25(3)~~~~100(4)~~~~80(5)~~60(6)

arr[top] <= arr[i]

30 <= 40 ✓

40 <= 20 ✗

20 <= 25 ✓

40 <= 25 ✗

25 <= 100 ✓

40 <= 100 ✓

100 <= 80 ✗

80 <= 60 ✗

⇒ HashMap (very useful)

↳ it store a pair of key and value

possibility:-

| | key | value |
|---|---------|---------|
| ↳ | Integer | Integer |
| ↳ | Integer | String |
| ↳ | String | Integer |
| ↳ | Integer | AL |
| ↳ | Integer | array |
| ↳ | : | |
| | etc. | |

ex: String, Integer

| | | |
|---|------------------|----------------|
| { | "Aus" | , 200 |
| | "Z" | , 150 |
| | "India" | , 500 |
| | "Pak" | , 0 |
| | "Srilanka" | , 50 |
| | "Pale" | , -10 |

Important properties of HM

- ↳ Store key/value pair
- ↳ It doesn't maintain an order (unorganised)
- ↳ In case of duplicate key, then new pair will overwrite previous value

Syntax

```
HashMap< KeyDataType, valueDataType> map = new HashMap<>();
```

```
→ HashMap< String, Integer> map = new HashMap<>();
```

Inbuilt functions

key value

```
map.put("Bharat", 500);  
map.put("Sri Lanka", 50);  
map.put("Sri Lanka", 100);  
map.remove("Sri Lanka")  
key
```

map

```
"Bharat" → 500  
"Sri Lanka" → 100
```

(10)

| <u>decimal</u> | | <u>binary (2)</u> |
|----------------|---|-------------------|
| 0 | → | 0 |
| 1 | → | 1 |
| 2 | → | 10 |
| 3 | → | 11 |
| 4 | → | 100 |
| 5 | → | 101 |
| 6 | → | 110 |
| 7 | → | 111 |
| 8 | → | 1000 |
| 9 | → | 1001 |

space