HW_Next greater element on left 1

arr = [3, 1, 2, 9, 4, 7]
(0) (1) (2) (3) (4) (5)

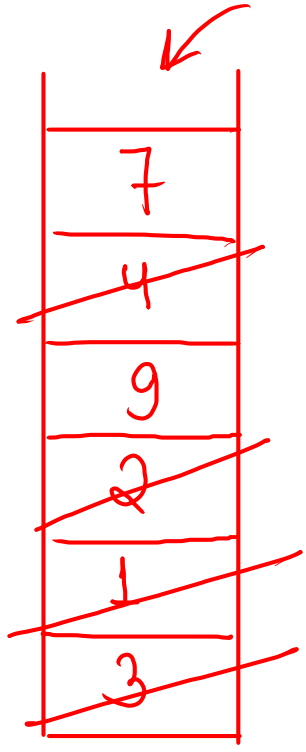
ans = [-1, 3, 3, -1, 9, 9]

arr = [3 , 1 , 2 , 9 , 4 , 7]

0 1 2 3 4 5

Ex:-

dry run



Ex:-

arr = [12, 8, 7, 6, 5, 4, 9, 8, 7]

ans = [-1, 12, 8, 7, 6, 5, 12, 9, 8]

Note:- after destroying all possible elements, whichever no. is present on top of stack that no. is your ans

psudo code

1) declare stack and array ans

2) loop from start to end

2.1) loop until peek <= curr
st.pop()

2.2) ans[i] = st.peek()

2.3) st.push(arr[i]);

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    int[] ans = greaterElementOnLeft(arr, n);
    for (int i: ans) {
        System.out.print(i + " ");
    }
}
public static int[] greaterElementOnLeft(int[] arr, int n) {
    Stack<Integer> st = new Stack<>();
    int[] ans = new int[n];
    for (int i = 0; i < n; i++) {
        while ( st.size() > 0 && st.peek() <= arr[i] ) {
            st.pop();
        }
        if (st.size() > 0) {
            ans[i] = st.peek();
        } else {
            ans[i] = -1;
        }
        st.push(arr[i]);
    }
    return ans;
}
```

T.C = O(N)

N = size of arr

Imp

1) Greater element on left side

2) Greater element on right side :- just reverse the loop

3) Smaller element on left side :- change cond" inside inner loop.

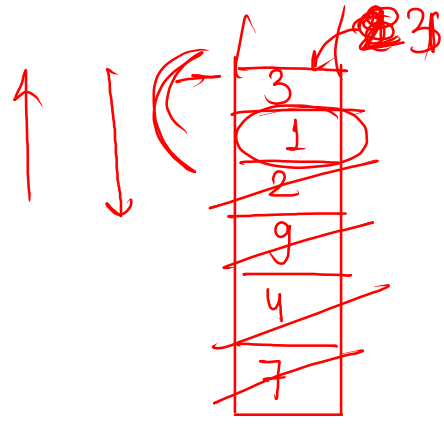
4) Smaller element on right side :- use both

Next Smaller Element To The Right

```java
public static int[] greaterElementOnLeft(int[] arr, int n) {
    Stack<Integer> st = new Stack<>();
    int[] ans = new int[n];
    for (int i = n - 1; i >= 0; i--) {
        while ( st.size() > 0 && st.peek() >= arr[i] ) {
            st.pop();
        }
        if (st.size() > 0) {
            ans[i] = st.peek();
        } else {
            ans[i] = -1;
        }
        st.push(arr[i]);
    }
    return ans;
}
```

arr = [3, 1, 2, 9, 4, 7]

ans = [-1, -1, -1, 4, -1, -1]



⇒ Hashmap [every operation in HM is having T.C constant]

→ actually :- Amotized O(1)

structure

hashmap

key → value

"India" → 400
"England" → 220
"SriLanka" → 170
"Aus" → 17
"Pak" → -10
"aus" → 170

Key (String)	Value (Integer)
Integer	Integer
Boolean	Boolean
Character	Character
String	String
Double	Double
Float	Float
Long	Long
	ArrayList
	Stack
	Queue
	PriorityQueue
	Hashmap

⇒ Imp notes for hashmap

1) all keys are always unique
2) keys are case sensitive
3) values can be repeated
4) if same key is added again in hashmap then previous pair will be overrided
5) data is unorganised (no indexing)

Syntax

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

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

Inbuilt

- `map.put(key, value);` // to add a pair in HM
- `map.get(key);` // return value of given key
- `map.remove(key);` // to delete entire pair

(Evergreen fn) :- [`map.size()`, `map.isEmpty()`]

↳ map.containsKey(key); // true if key is present in HM, false otherwise

↳ map.containsValue(value); // true if value is present in HM, false otherwise

Note:- all these tasks are completing in O(1) time.

Ex:-

String vs Integer

"abc"	3
"efg"	8
"EFG"	7

```
map.put("abc", 3);
map.put(4, "efg");   error
map.put("efg", 4);
map.put("EFG", 7);
map.put("efg", 8);
map.get("efg");      // 8
map.isEmpty();       // false
map.get("xyz");      // error, null
map.containsKey("abc");//true
map.containsValue(27); //false
```