```
                qmax.pollLast();
            }
            qmax.addLast(i);
            if (qmax.peekFirst() == i - w) {
                qmax.pollFirst();
            }
            if (i >= w - 1) {
                res[index++] = arr[qmax.peekFirst()];
            }
        }
        return res;
    }
```

构造数组的 MaxTree

【题目】

定义二叉树节点如下：

```
public class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
            this.value = data;
        }
}
```

一个数组的 MaxTree 定义如下。

- 数组必须没有重复元素。
- MaxTree 是一棵二叉树，数组的每一个值对应一个二叉树节点。
- 包括 MaxTree 树在内且在其中的每一棵子树上，值最大的节点都是树的头。

给定一个没有重复元素的数组 arr，写出生成这个数组的 MaxTree 的函数，要求如果数组长度为 N，则时间复杂度为 $O(N)$、额外空间复杂度为 $O(N)$。

【难度】

校 ★★★☆

【解答】

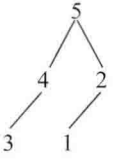
下面举例说明如何在满足时间和空间复杂度的要求下生成 MaxTree。

```
arr = {3, 4, 5, 1, 2}
3 的左边第一个比 3 大的数：无          3 的右边第一个比 3 大的数：4
4 的左边第一个比 4 大的数：无          4 的右边第一个比 4 大的数：5
5 的左边第一个比 5 大的数：无          5 的右边第一个比 5 大的数：无
1 的左边第一个比 1 大的数：5          1 的右边第一个比 1 大的数：2
2 的左边第一个比 2 大的数：5          2 的右边第一个比 2 大的数：无
```

以下列原则来建立这棵树：

- 每一个数的父节点是它左边第一个比它大的数和它右边第一个比它大的数中，较小的那个。
- 如果一个数左边没有比它大的数，右边也没有。也就是说，这个数是整个数组的最大值，那么这个数是 MaxTree 的头节点。

那么 3, 4, 5, 1, 2 的 MaxTree 如下：



为什么通过这个方法能够正确地生成 MaxTree 呢？我们需要给出证明，证明分为如下两步。

1. 通过这个方法，所有的数能生成一棵树，这棵树可能不是二叉树，但肯定是一棵树，而不是多棵树（森林）。

我们知道，在数组中的所有数都不同，而一个较小的数肯定会以一个比自己大的数作为父节点，那么最终所有的数向上找都会找到数组中的最大值，所以它们会有一个共同的头。证明完毕。

2. 通过这个方法，所有的数最多都只有两个孩子。也就是说，这棵树可以用二叉树表示，而不需要多叉树。

要想证明这个问题，只需证明任何一个数在单独一侧，孩子数量都不可能超过 1 个即可。

假设 a 这个数在单独一侧有 2 个孩子，不妨设在右侧。假设这两个孩子一个是 k1，另一个是 k2，即

```
…a…k1…k2…
```

因为 a 是 k1 和 k2 的父，所以 a>k1，a>k2。根据题意，k1 和 k2 不相等，所以 k1 和 k2 可以分出大小，先假设 k1 是较小的，k2 是较大的：

那么 k1 可能会以 k2 为父节点，而绝对不会以 a 为父节点，因为根据我们的方法，每一个数的父节点是它左边第一个比它大的数和它右边第一个比它大的数中较小的那个，又有 a>k2。

再假设 k2 是较小的，k1 是较大的：

那么 k2 可能会以 k1 为父节点，也绝对不会以 a 为父节点，因为根据我们的方法，k1 才可能是 k2 左边第一个遇到的比 k2 大的数，而绝对不会轮到 a。

总之，k1 和 k2 肯定有一个不是 a 的孩子。

所以，任何一个数的单独一侧，其孩子数量都不可能超过 1 个，最多只会有 1 个。进而我们知道，任何一个数最多会有 2 个孩子，而不会有更多。

证明完毕。

以上证明了该方法是有效的，那么如何尽可能快地找到每一个数左右两边第一个比它大的数呢？利用栈。

找每个数左边第一个比它大的数，从左到右遍历每个数，栈中保持递减序列，新来的数不停地利用 Pop 出栈顶，直到栈顶比新数大或没有数。

以[3,1,2]为例，首先 3 入栈，接下来 1 比 3 小，无须 pop 出 3，1 入栈，并且确定了 1 往左第一个比它大的数为 3。接下来 2 比 1 大，1 出栈，2 比 3 小，2 入栈，并且确定了 2 往左第一个比它大的数为 3。

用同样的方法可以求得每个数往右第一个比它大的数。

具体请参看如下代码中的 getMaxTree 方法。

```
public Node getMaxTree(int[] arr) {
    Node[] nArr = new Node[arr.length];
    for (int i = 0; i != arr.length; i++) {
            nArr[i] = new Node(arr[i]);
    }
    Stack<Node> stack = new Stack<Node>();
    HashMap<Node, Node> lBigMap = new HashMap<Node, Node>();
    HashMap<Node, Node> rBigMap = new HashMap<Node, Node>();
    for (int i = 0; i != nArr.length; i++) {
            Node curNode = nArr[i];
            while ((!stack.isEmpty()) && stack.peek().value < curNode.value) {
                    popStackSetMap(stack, lBigMap);
            }
            stack.push(curNode);
    }
```

```
    while (!stack.isEmpty()) {
        popStackSetMap(stack, lBigMap);
    }
    for (int i = nArr.length - 1; i != -1; i--) {
            Node curNode = nArr[i];
            while ((!stack.isEmpty()) && stack.peek().value < curNode.value) {
                    popStackSetMap(stack, rBigMap);
            }
            stack.push(curNode);
    }
    while (!stack.isEmpty()) {
            popStackSetMap(stack, rBigMap);
    }
    Node head = null;
    for (int i = 0; i != nArr.length; i++) {
            Node curNode = nArr[i];
            Node left = lBigMap.get(curNode);
            Node right = rBigMap.get(curNode);
            if (left == null && right == null) {
                    head = curNode;
            } else if (left == null) {
                    if (right.left == null) {
                            right.left = curNode;
                    } else {
                            right.right = curNode;
                    }
            } else if (right == null) {
                    if (left.left == null) {
                            left.left = curNode;
                    } else {
                            left.right = curNode;
                    }
            } else {
                    Node parent = left.value < right.value ? left : right;
                    if (parent.left == null) {
                            parent.left = curNode;
                    } else {
                            parent.right = curNode;
                    }
            }
    }
    return head;
}


public void popStackSetMap(Stack<Node> stack, HashMap<Node, Node> map) {
    Node popNode = stack.pop();
    if (stack.isEmpty()) {
            map.put(popNode, null);
    } else {
            map.put(popNode, stack.peek());
    }
}
```