> 双指针风骚起来，也是无敌

977.有序数组的平方

https://leetcode-cn.com/problems/squares-of-a-sorted-array/

给你一个按 非递减顺序 排序的整数数组 nums，返回 每个数字的平方 组成的新数组，要求也按 非递减顺序 排序。

set肯定不行，不能删除元素

示例 1： 输入：nums = [-4,-1,0,3,10] 输出：[0,1,9,16,100] 解释：平方后，数组变为 [16,1,0,9,100]，排序后，数组变为 [0,1,9,16,100]

示例 2： 输入：nums = [-7,-3,2,3,11] 输出：[4,9,9,49,121]

思路

暴力排序

最直观的相反，莫过于：每个数平方之后，排个序，美滋滋，代码如下：

```
class Solution {
public:
    vector<int> sortedSquares(vector<int>& A) {
        for (int i = 0; i < A.size(); i++) {
```

```
            A[i] *= A[i];
        }
        sort(A.begin(), A.end()); // 快速排序
        return A;
    }
};
```

这个时间复杂度是 O(n + nlogn)， 可以说是O(nlogn)的时间复杂度，但为了和下面双指针法算法时间复杂度有鲜明对比，我记为 O(n + nlogn)。

空间复杂度：O(1)

双指针法

数组其实是有序的， 只不过负数平方之后可能成为最大数了。

那么数组平方的最大值就在数组的两端，不是最左边就是最右边，不可能是中间。

此时可以考虑双指针法了，i指向起始位置，j指向终止位置。

定义一个新数组result，和A数组一样的大小，让k指向result数组终止位置。

如果 `A[i] * A[i] < A[j] * A[j]` 那么 `result[k--] = A[j] * A[j];` 。

如果 `A[i] * A[i] >= A[j] * A[j]` 那么 `result[k--] = A[i] * A[i];` 。

如动画所示：

不难写出如下代码：

```
class Solution {
public:
    vector<int> sortedSquares(vector<int>& A) {
        int k = A.size() - 1;
        vector<int> result(A.size(), 0);
        for (int i = 0, j = A.size() - 1; i <= j;) { // 注意这里要i <= j，因为最后要处理两个元素
            if (A[i] * A[i] < A[j] * A[j])  {
                result[k--] = A[j] * A[j];
                j--;
            }
            else {
                result[k--] = A[i] * A[i];
                i++;
            }
        }
        return result;
    }
};
```

此时的时间复杂度为O(n)，相对于暴力排序的解法O(n + nlogn)还是提升不少的。

空间复杂度：O(n)

这里还是说一下，大家不必太在意leetcode上执行用时，打败多少多少用户，这个就是一个玩具，非常不准确。

做题的时候自己能分析出来时间复杂度就可以了，至于leetcode上执行用时，大概看一下就行，只要达到最优的时间复杂度就可以了，

一样的代码多提交几次可能就击败百分之百了.....

其他语言版本

Java：

```
class Solution {
    public int[] sortedSquares(int[] nums) {
        int right = nums.length - 1;
        int left = 0;
        int[] result = new int[nums.length];
        int index = result.length - 1;
        while (left <= right) {
            if (nums[left] * nums[left] > nums[right] * nums[right]) {
                result[index--] = nums[left] * nums[left];
                ++left;
            } else {
                result[index--] = nums[right] * nums[right];
                --right;
            }
        }
        return result;
    }
}
```

```
class Solution {
    public int[] sortedSquares(int[] nums) {
        int l = 0;
        int r = nums.length - 1;
        int[] res = new int[nums.length];
        int j = nums.length - 1;
        while(l <= r){
            if(nums[l] * nums[l] > nums[r] * nums[r]){
                res[j--] = nums[l] * nums[l++];
            }else{
                res[j--] = nums[r] * nums[r--];
            }
        }
        return res;
    }
}
```

Python：

```
class Solution:
    def sortedSquares(self, nums: List[int]) -> List[int]:
```

```
        n = len(nums)
        i,j,k = 0,n - 1,n - 1
        ans = [-1] * n
        while i <= j:
            lm = nums[i] ** 2
            rm = nums[j] ** 2
            if lm > rm:
                ans[k] = lm
                i += 1
            else:
                ans[k] = rm
                j -= 1
            k -= 1
        return ans
```

Go：

```
func sortedSquares(nums []int) []int {
        n := len(nums)
        i, j, k := 0, n-1, n-1
        ans := make([]int, n)
        for i <= j {
                lm, rm := nums[i]*nums[i], nums[j]*nums[j]
                if lm > rm {
                        ans[k] = lm
                        i++
                } else {
                        ans[k] = rm
                        j--
                }
                k--
        }
        return ans
}
```

Rust

```
impl Solution {
    pub fn sorted_squares(nums: Vec<i32>) -> Vec<i32> {
        let n = nums.len();
        let (mut i,mut j,mut k) = (0,n - 1,n- 1);
        let mut ans = vec![0;n];
        while i <= j{
            if nums[i] * nums[i] < nums[j] * nums[j] {
                ans[k] = nums[j] * nums[j];
                j -= 1;
            }else{
                ans[k] = nums[i] * nums[i];
                i += 1;
            }
            k -= 1;
        }
        ans
    }
}
```

Javascript：

```
/**
 * @desc two pointers solution
 * @link https://leetcode-cn.com/problems/squares-of-a-sorted-array/
 * @param nums Array  e.g. [-4,-1,0,3,10]
 * @return {array}    e.g. [0,1,9,16,100]
 */
const sortedSquares = function (nums) {
    let res = []
    for (let i = 0, j = nums.length - 1; i <= j;) {
      const left = Math.abs(nums[i])
      const right = Math.abs(nums[j])
      if (right > left) {
        // push element to the front of the array
        res.unshift(right * right)
        j--
      } else {
        res.unshift(left * left)
        i++
      }
    }
    return res
  }
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录