912. Sort an Array

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in `O(nlog(n))` time complexity and with the smallest space complexity possible.

Example 1:

```
Input: nums = [5,2,3,1]
Output: [1,2,3,5]
Explanation: After sorting the array, the positions of some numbers are not changed
(for example, 2 and 3), while the positions of other numbers are changed (for
example, 1 and 5).
```

Example 2:

```
Input: nums = [5,1,1,2,0,0]
Output: [0,0,1,1,2,5]
Explanation: Note that the values of nums are not necessairly unique.
```

Constraints:

- `1 <= nums.length <= 5 * 10`4
- `-5 * 10`4 `<= nums[i] <= 5 * 10`4

912. Sort an Array

```
/*
    priority_queue (min heap)
    Time complexity: O(nlogn)
    Extra space complexity: O(n)
*/
class Solution {
public:
    std::vector<int> sortArray(std::vector<int>& nums) {
        std::priority_queue<int,std::vector<int>,std::greater<int>> q;
        for(auto& e: nums) q.push(e);
        int i=0;
        while(!q.empty()){
            nums[i++]=q.top();
            q.pop();
        }
        return nums;
    }
};
```

912. Sort an Array

```
/*
    Custom min heap (without heapify method)
    Time complexity: O(nlogn)
    Extra space complexity: O(n)
*/
class Solution {
    public:
        class MinHeap{
            public:
                std::vector<int> A;
            public:
                // Time complexity: O(logn)
                void push(int val){
                    A.push_back(val);
                    int i=A.size()-1;
                    while(i>0 && A[i]<A[(i-1)/2]){
                        std::swap(A[i],A[(i-1)/2]);
                        i=(i-1)/2;
                    }
                }

                // Time complexity: O(1)
                bool is_empty(){return A.size()==0;}

                // Time complexity: O(1)
                int get_index_of_mininum_child(int i){
                    if(2*i+2<A.size()){
                        if(A[2*i+1]<A[2*i+2]) return 2*i+1;
                        return 2*i+2;
                    }
                    if(2*i+1<A.size()) return 2*i+1;


                    return -1; // Leaf node reached
                }
                // Time complexity: O(logn)
                void pop(){
                    if(is_empty()) return;
                    std::swap(A[0],A[A.size()-1]);
                    A.pop_back();

                    if(is_empty()) return;

                    int i=0;
                    int min_id=get_index_of_mininum_child(i);
                    while(min_id!=-1 && A[min_id]<A[i]){
                        std::swap(A[min_id],A[i]);
                        i=min_id;
                        min_id=get_index_of_mininum_child(i);
                    }
                }

                // Time complexity: O(1)
                int top(){
                    if(is_empty()) {
                        std::cout<<"Min heap is empty\n";
                        return INT_MAX;
                    }
                    return A[0];
                }
        };
    public:
        std::vector<int> sortArray(std::vector<int>& nums){
            MinHeap min_heap=MinHeap();
            // Time complexity: O(nlogn)
            for(auto& val: nums) min_heap.push(val);

            // Time complexity: O(nlogn)
            int i=0;
            while(!min_heap.is_empty()){
                nums[i++]=min_heap.top();
                min_heap.pop();
            }
            return nums;
        }
};
```

912. Sort an Array

```
/*
    Custom min heap (with heapify method)
    Time complexity: O(nlogn)
    Extra space complexity: O(n)
*/
class Solution {
    public:
        class MinHeap{
            public:
                std::vector<int> A;
            public:
                // Time complexity: O(1)
                bool is_empty(){return A.size()==0;}

                // Time complexity: O(1)
                int get_index_of_mininum_child(int i,int n){
                    if(2*i+2<n){
                        if(A[2*i+1]<A[2*i+2]) return 2*i+1;
                        return 2*i+2;
                    }
                    if(2*i+1<n) return 2*i+1;


                    return -1; // leaf node reached
                }
                // Time complexity: O(logn)
                void heapify(int i,int n){
                    int min_id=get_index_of_mininum_child(i,n);
                    while(min_id!=-1 && A[min_id]<A[i]){
                        std::swap(A[i],A[min_id]);
                        i=min_id;
                        min_id=get_index_of_mininum_child(i,n);
                    }
                }

        };
```

```
  public:
        std::vector<int> sortArray(std::vector<int>& nums){
            MinHeap min_heap=MinHeap();

            min_heap.A=nums;

            int n=nums.size();

            for(int i=n/2-1;i>=0;i--) min_heap.heapify(i,n);

            for(int i=n-1;i>=0;i--) {
               std::swap(min_heap.A[0],min_heap.A[i]);
                min_heap.heapify(0,i);
            }

            for(int i=n-1;i>=0;--i) nums[n-i-1]=min_heap.A[i];

            return nums;
        }
};
```