11. Container With Most Water

Medium

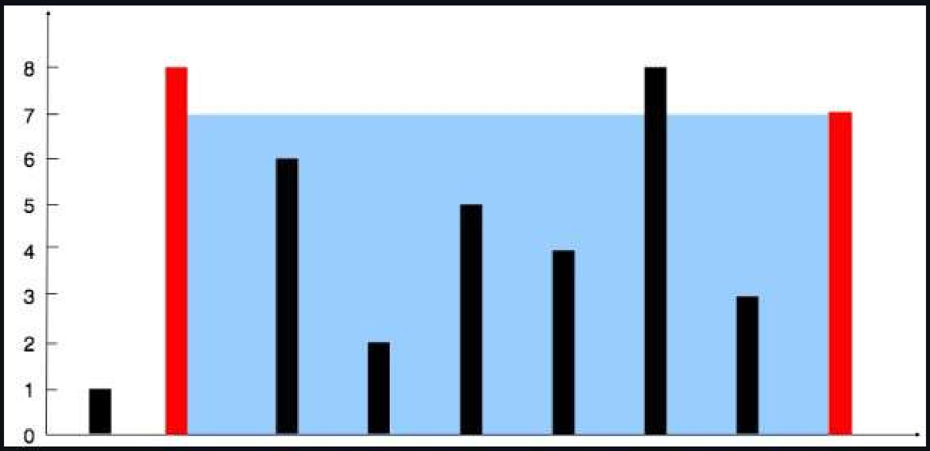
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `ith` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

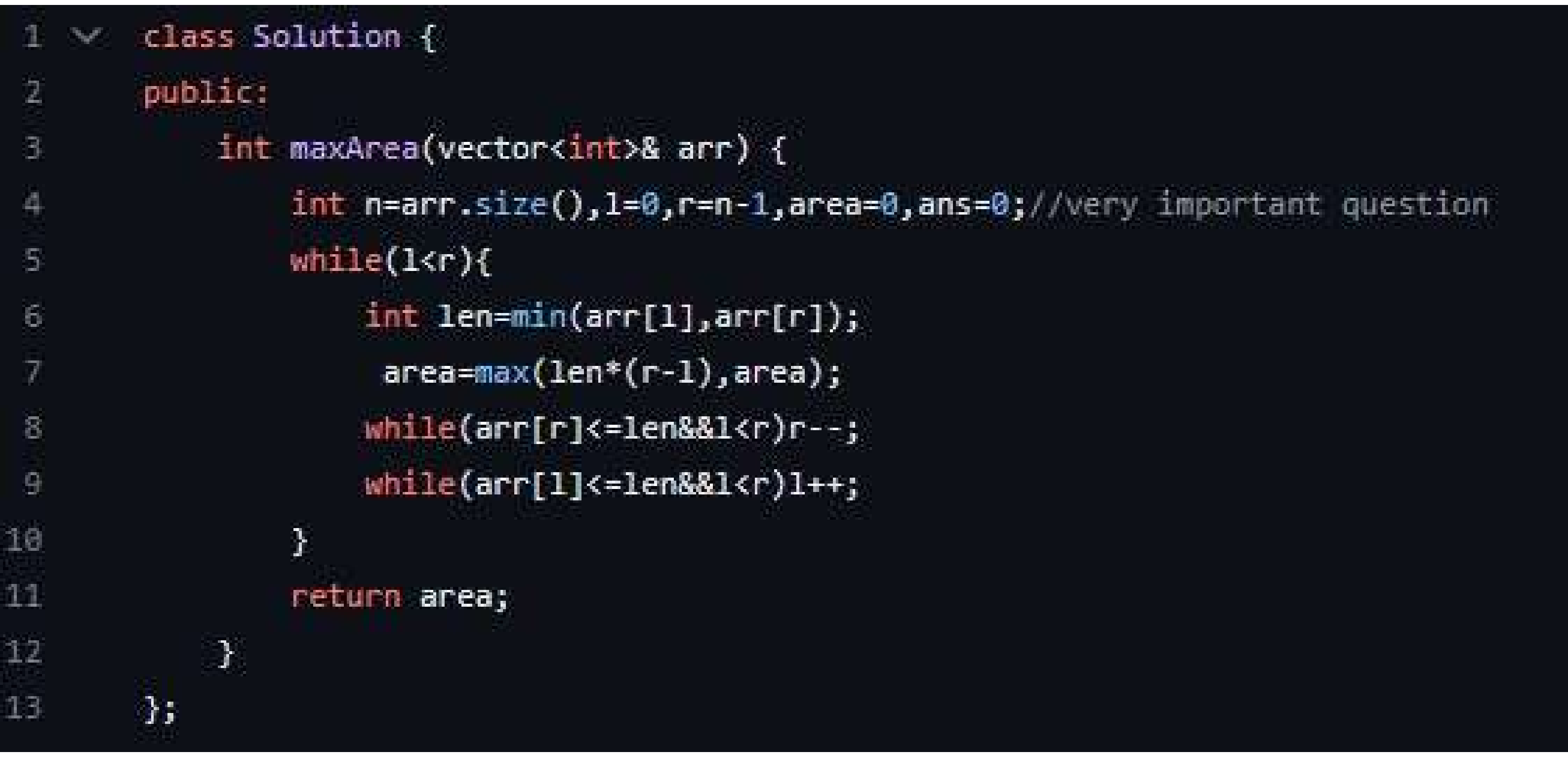
Return *the maximum amount of water a container can store.*

Notice that you may not slant the container.

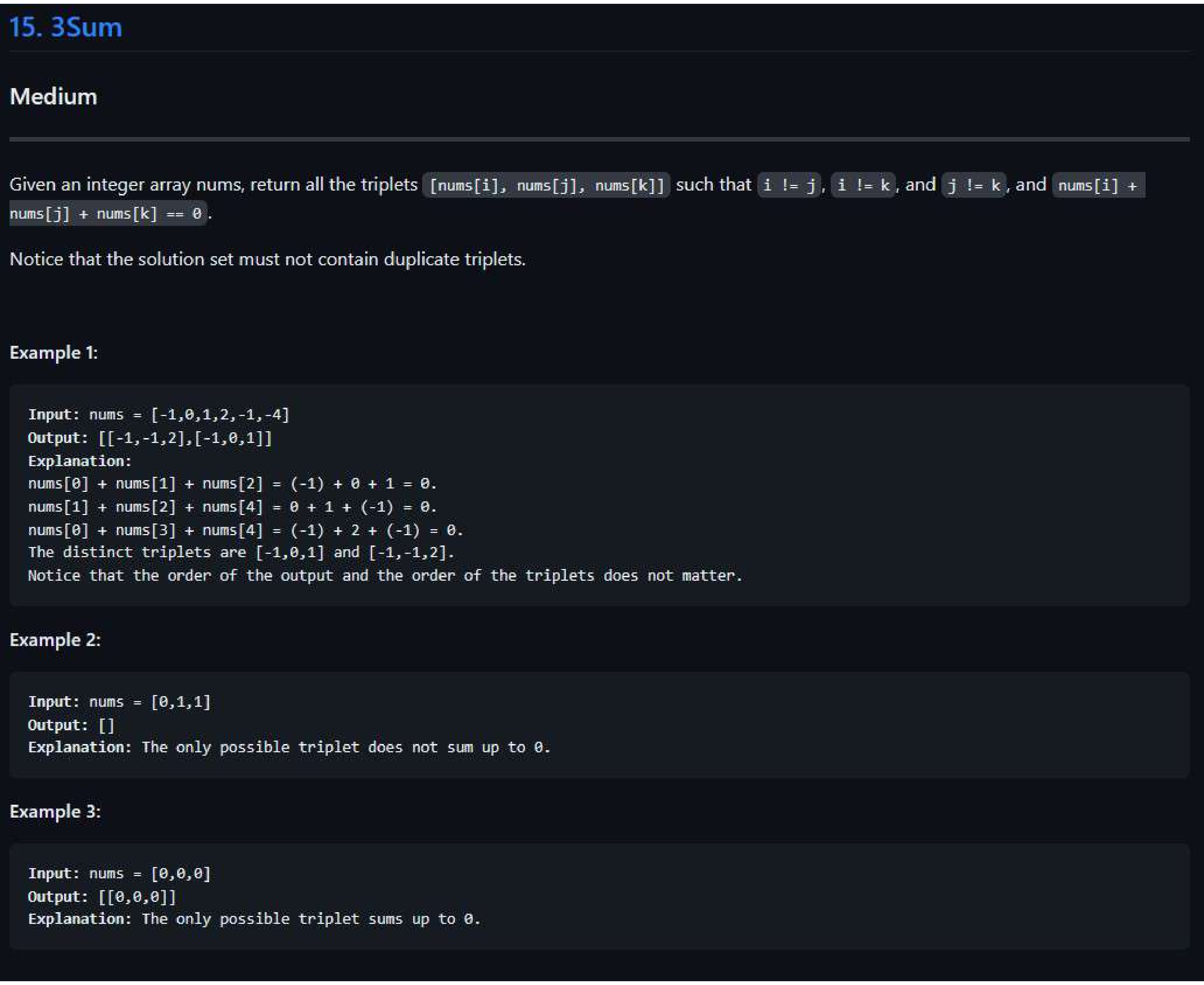
Example 1:



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section
```



```cpp
class Solution {
public:
    int maxArea(vector<int>& arr) {
        int n=arr.size(),l=0,r=n-1,area=0,ans=0;//very important question
        while(l<r){
            int len=min(arr[l],arr[r]);
             area=max(len*(r-l),area);
            while(arr[r]<=len&&l<r)r--;
            while(arr[l]<=len&&l<r)l++;
        }
        return area;
    }
};
```



15. 3Sum

Medium

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

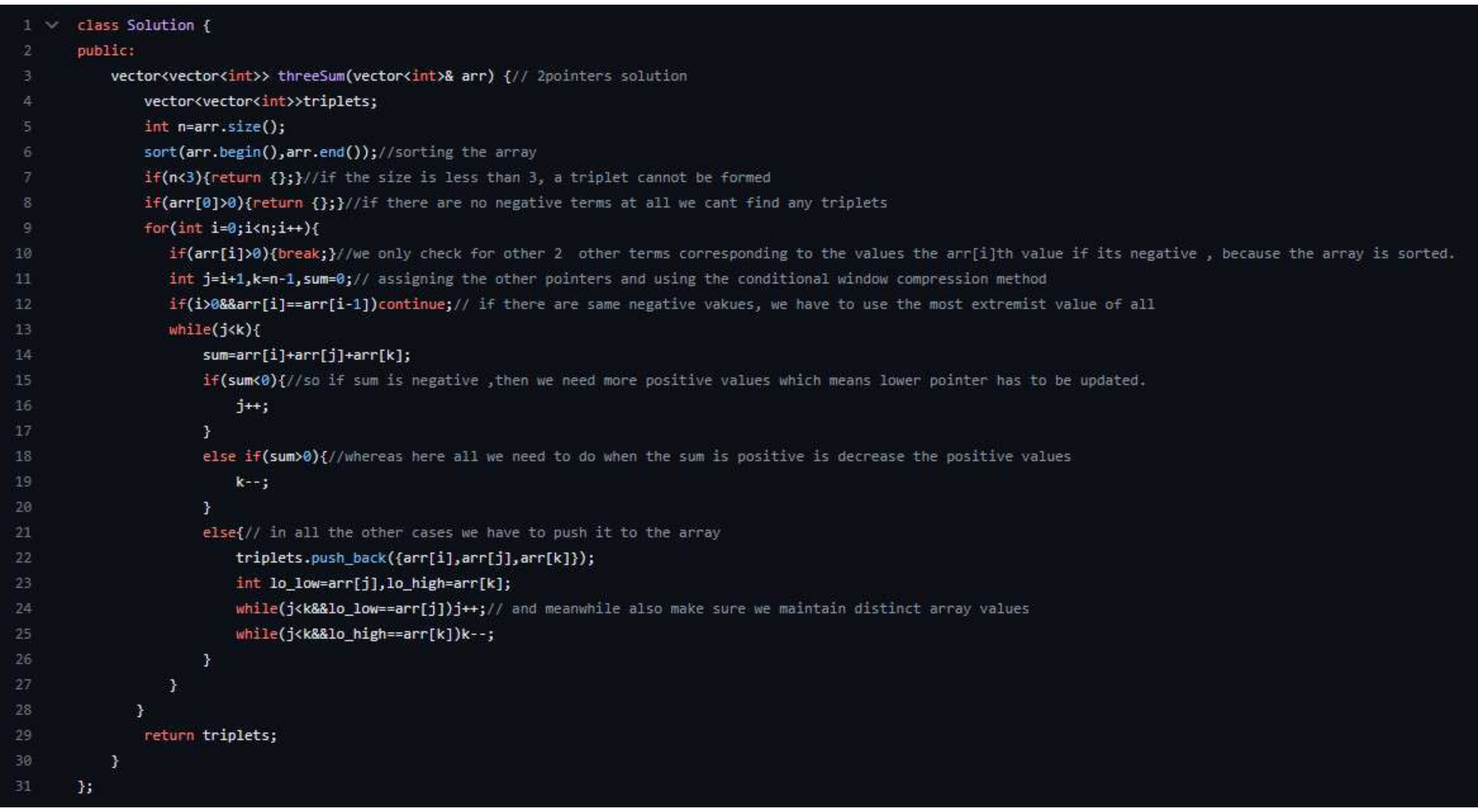
```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
```

Example 2:

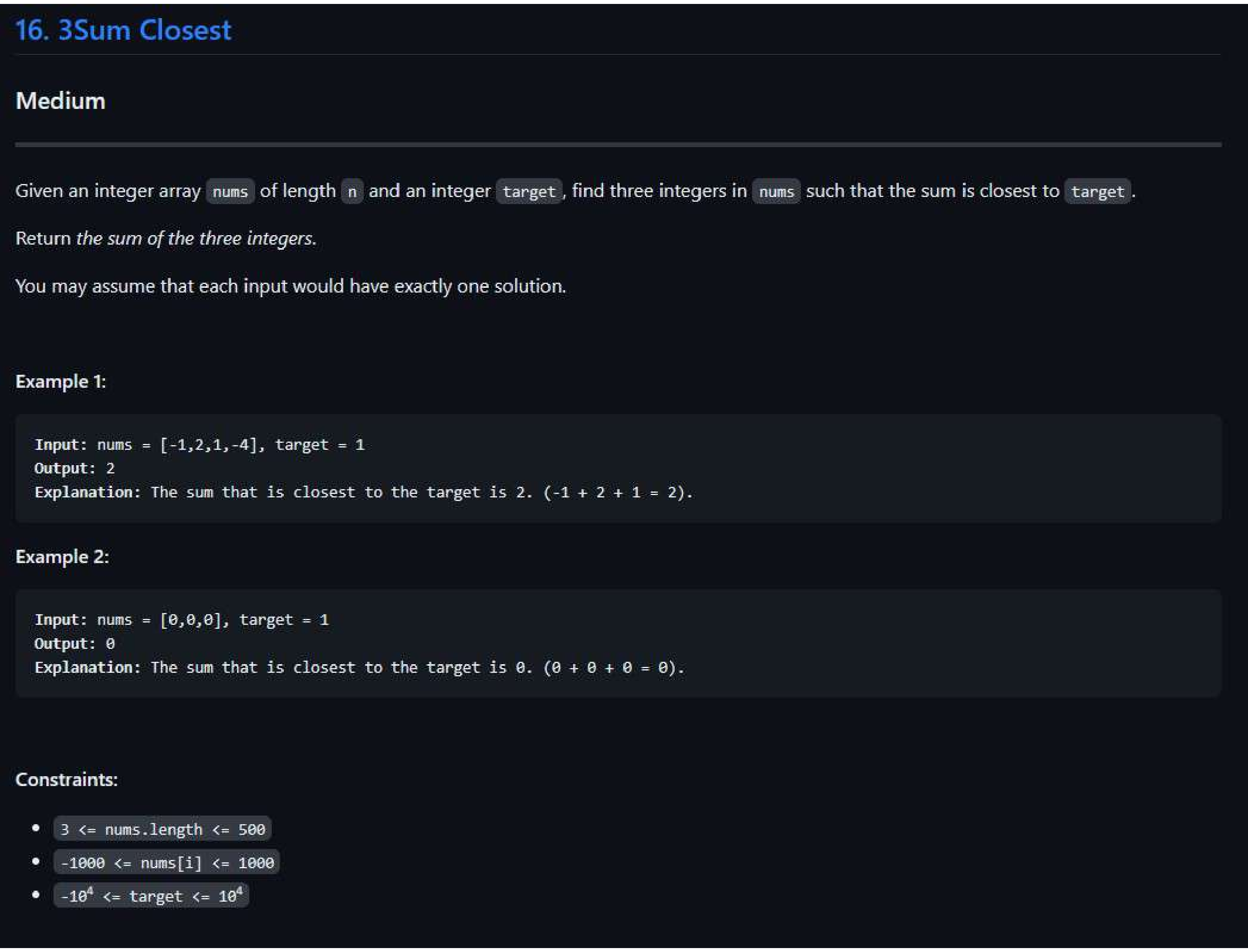
```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```



```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& arr) {// 2pointers solution
        vector<vector<int>>triplets;
        int n=arr.size();
        sort(arr.begin(),arr.end());//sorting the array
        if(n<3){return {};}//if the size is less than 3, a triplet cannot be formed
        if(arr[0]>0){return {};}//if there are no negative terms at all we cant find any triplets
        for(int i=0;i<n;i++){
           if(arr[i]>0){break;}//we only check for other 2  other terms corresponding to the values the arr[i]th value if its negative , because the array is sorted.
           int j=i+1,k=n-1,sum=0;// assigning the other pointers and using the conditional window compression method
           if(i>0&&arr[i]==arr[i-1])continue;// if there are same negative vakues, we have to use the most extremist value of all
           while(j<k){
               sum=arr[i]+arr[j]+arr[k];
               if(sum<0){//so if sum is negative ,then we need more positive values which means lower pointer has to be updated.
                   j++;
               }
               else if(sum>0){//whereas here all we need to do when the sum is positive is decrease the positive values
                   k--;
               }
               else{// in all the other cases we have to push it to the array
                   triplets.push_back({arr[i],arr[j],arr[k]});
                   int lo_low=arr[j],lo_high=arr[k];
                   while(j<k&&lo_low==arr[j])j++;// and meanwhile also make sure we maintain distinct array values
                   while(j<k&&lo_high==arr[k])k--;
               }
           }
       }
        return triplets;
    }
};
```



16. 3Sum Closest

Medium

Given an integer array `nums` of length `n` and an integer `target`, find three integers in `nums` such that the sum is closest to `target`.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

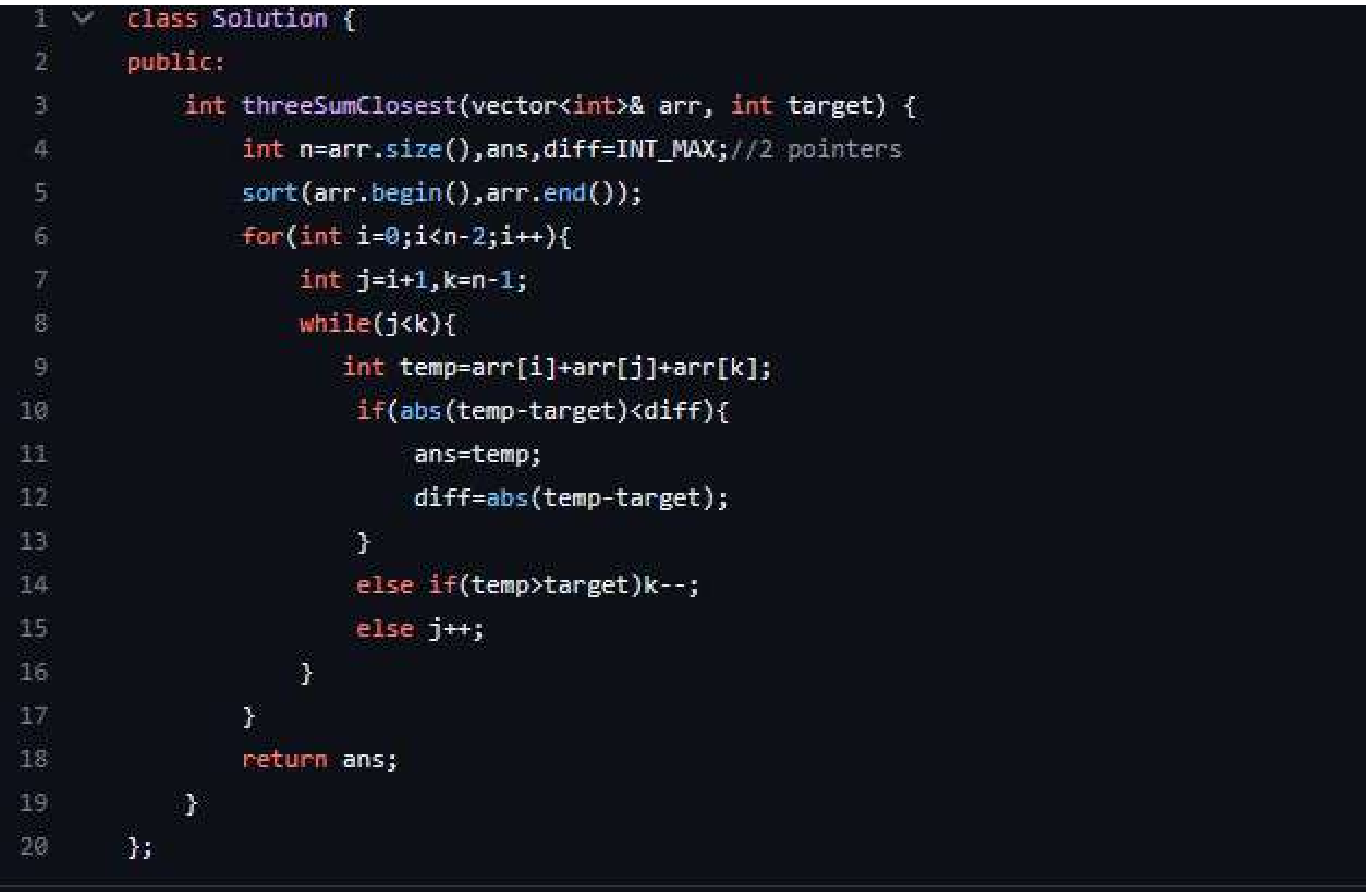
```
Input: nums = [-1,2,1,-4], target = 1
Output: 2
Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Example 2:

```
Input: nums = [0,0,0], target = 1
Output: 0
Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).
```

Constraints:

- `3 <= nums.length <= 500`
- `-1000 <= nums[i] <= 1000`
- -10^4 `<= target <=` 10^4



```cpp
class Solution {
public:
    int threeSumClosest(vector<int>& arr, int target) {
        int n=arr.size(),ans,diff=INT_MAX;//2 pointers
        sort(arr.begin(),arr.end());
        for(int i=0;i<n-2;i++){
            int j=i+1,k=n-1;
            while(j<k){
               int temp=arr[i]+arr[j]+arr[k];
                if(abs(temp-target)<diff){
                    ans=temp;
                    diff=abs(temp-target);
                }
                else if(temp>target)k--;
                else j++;
            }
        }
        return ans;
    }
};
```

18. 4Sum

Medium

Given an array `nums` of `n` integers, return *an array of all the* ***unique*** *quadruplets* `[nums[a], nums[b], nums[c], nums[d]]` such that:

- `0 <= a, b, c, d < n`
- `a`, `b`, `c`, and `d` are **distinct**.
- `nums[a] + nums[b] + nums[c] + nums[d] == target`

You may return the answer in **any order**.

Example 1:

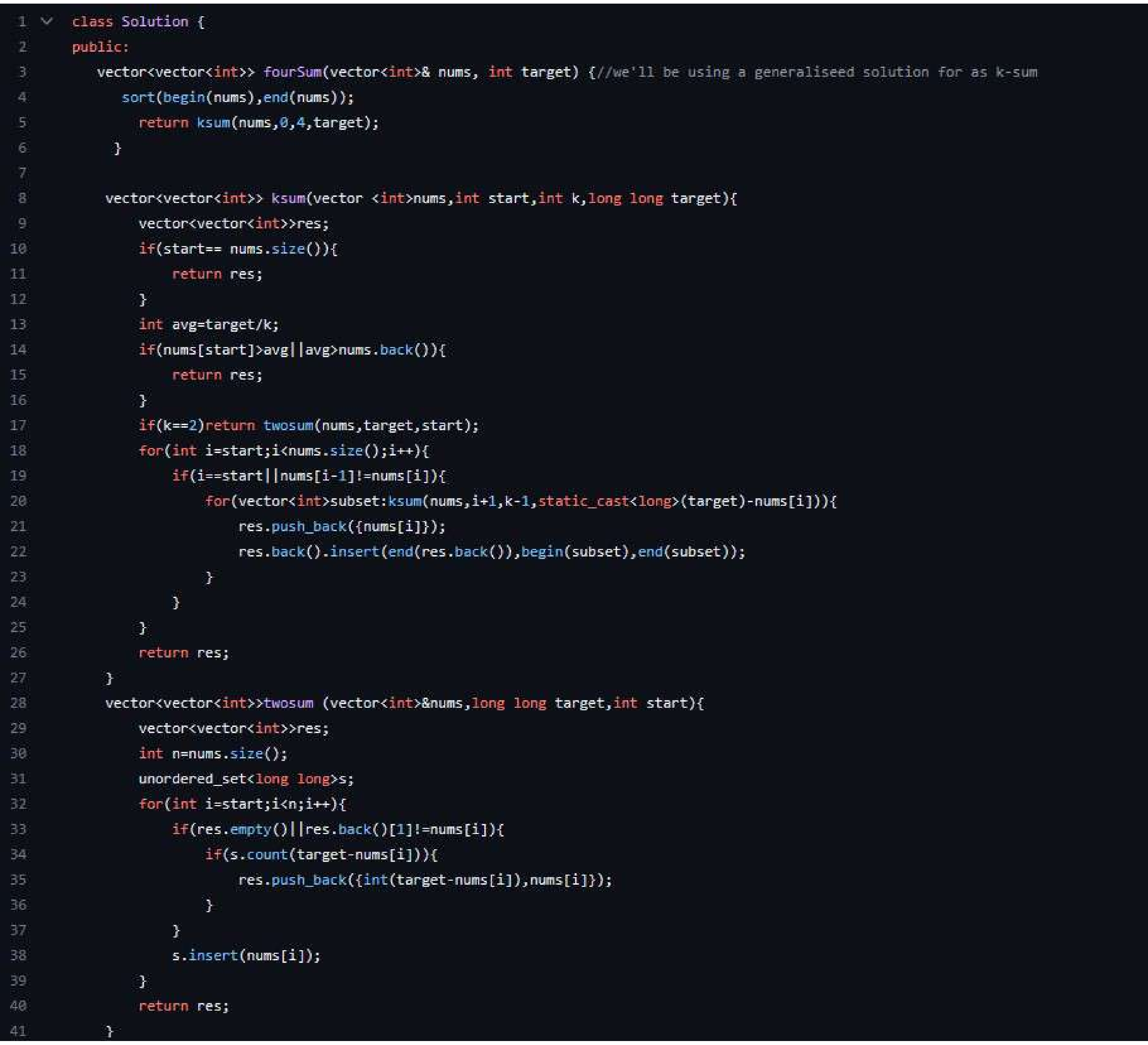
```
Input: nums = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
```

Example 2:

```
Input: nums = [2,2,2,2,2], target = 8
Output: [[2,2,2,2]]
```

Constraints:

- `1 <= nums.length <= 200`
- $-10^9 \leq \texttt{nums[i]} \leq 10^9$
- $-10^9 \leq \texttt{target} \leq 10^9$



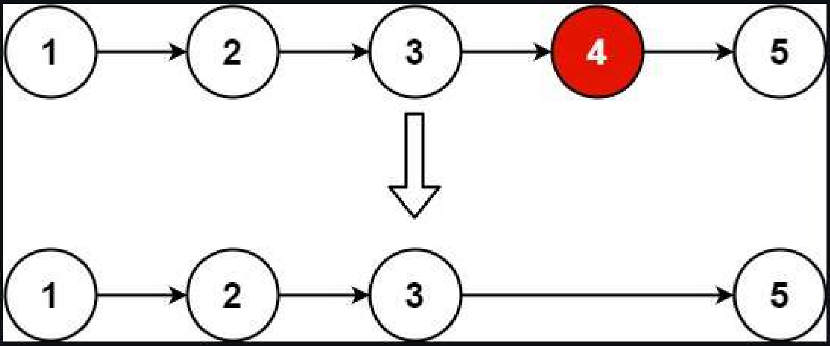
```cpp
class Solution {
public:
   vector<vector<int>> fourSum(vector<int>& nums, int target) {//we'll be using a generaliseed solution for as k-sum
      sort(begin(nums),end(nums));
        return ksum(nums,0,4,target);
     }

    vector<vector<int>> ksum(vector <int>nums,int start,int k,long long target){
        vector<vector<int>>res;
        if(start== nums.size()){
            return res;
        }
        int avg=target/k;
        if(nums[start]>avg||avg>nums.back()){
            return res;
        }
        if(k==2)return twosum(nums,target,start);
        for(int i=start;i<nums.size();i++){
            if(i==start||nums[i-1]!=nums[i]){
                for(vector<int>subset:ksum(nums,i+1,k-1,static_cast<long>(target)-nums[i])){
                    res.push_back({nums[i]});
                    res.back().insert(end(res.back()),begin(subset),end(subset));
                }
            }
        }
        return res;
    }
    vector<vector<int>>twosum (vector<int>&nums,long long target,int start){
        vector<vector<int>>res;
        int n=nums.size();
        unordered_set<long long>s;
        for(int i=start;i<n;i++){
            if(res.empty()||res.back()[1]!=nums[i]){
                if(s.count(target-nums[i])){
                    res.push_back({int(target-nums[i]),nums[i]});
                }
            }
            s.insert(nums[i]);
        }
        return res;
    }
```

19. Remove Nth Node From End of List

Medium

Given the `head` of a linked list, remove the `n^th` node from the end of the list and return its head.

Example 1:



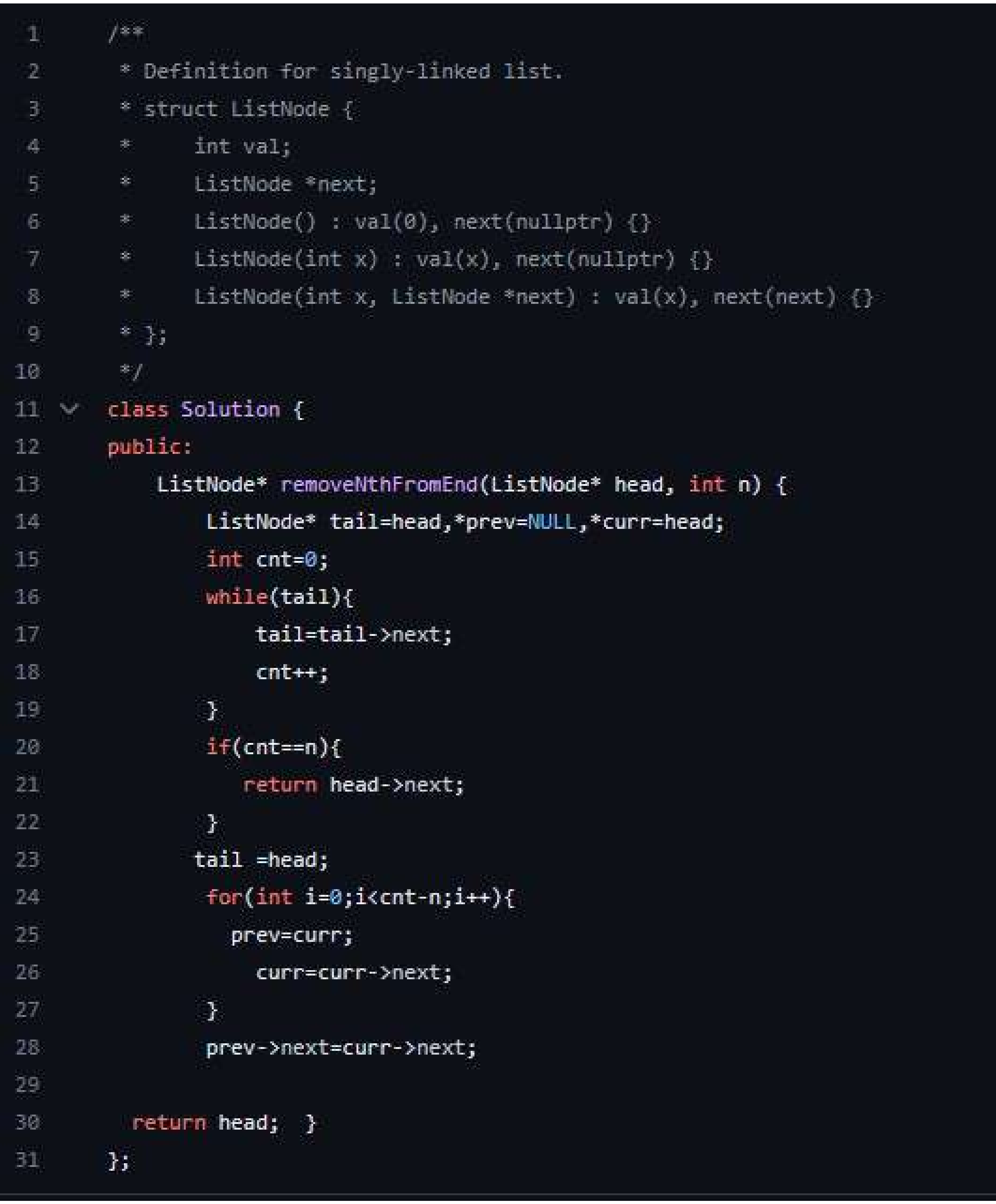
```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

Example 2:

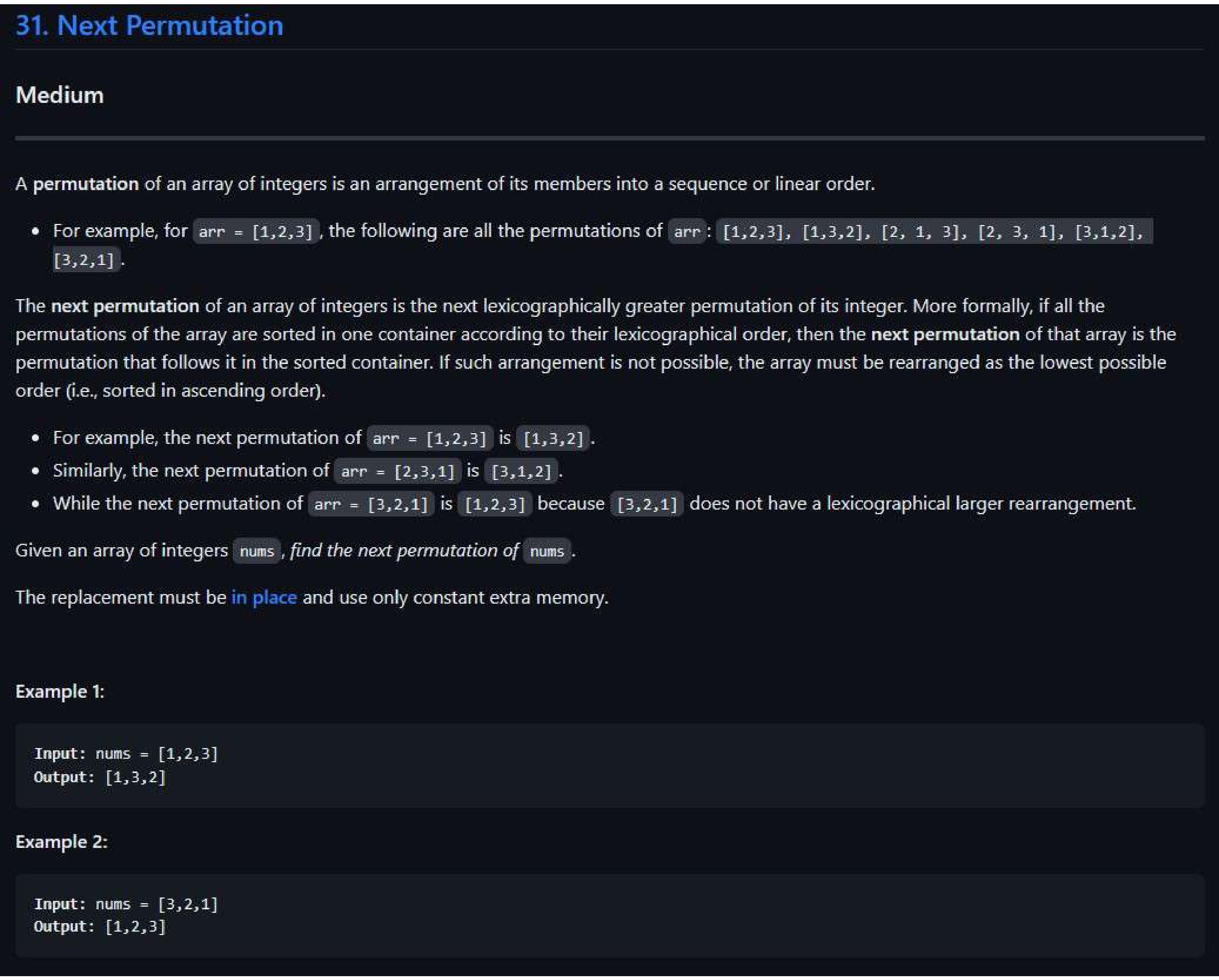
```
Input: head = [1], n = 1
Output: []
```

Example 3:

```
Input: head = [1,2], n = 1
Output: [1]
```



```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* tail=head,*prev=NULL,*curr=head;
        int cnt=0;
        while(tail){
            tail=tail->next;
            cnt++;
        }
        if(cnt==n){
           return head->next;
        }
       tail =head;
        for(int i=0;i<cnt-n;i++){
          prev=curr;
            curr=curr->next;
        }
        prev->next=curr->next;

  return head;  }
};
```



31. Next Permutation

Medium

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`.
- Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`.
- While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a lexicographical larger rearrangement.

Given an array of integers `nums`, *find the next permutation of* `nums`.

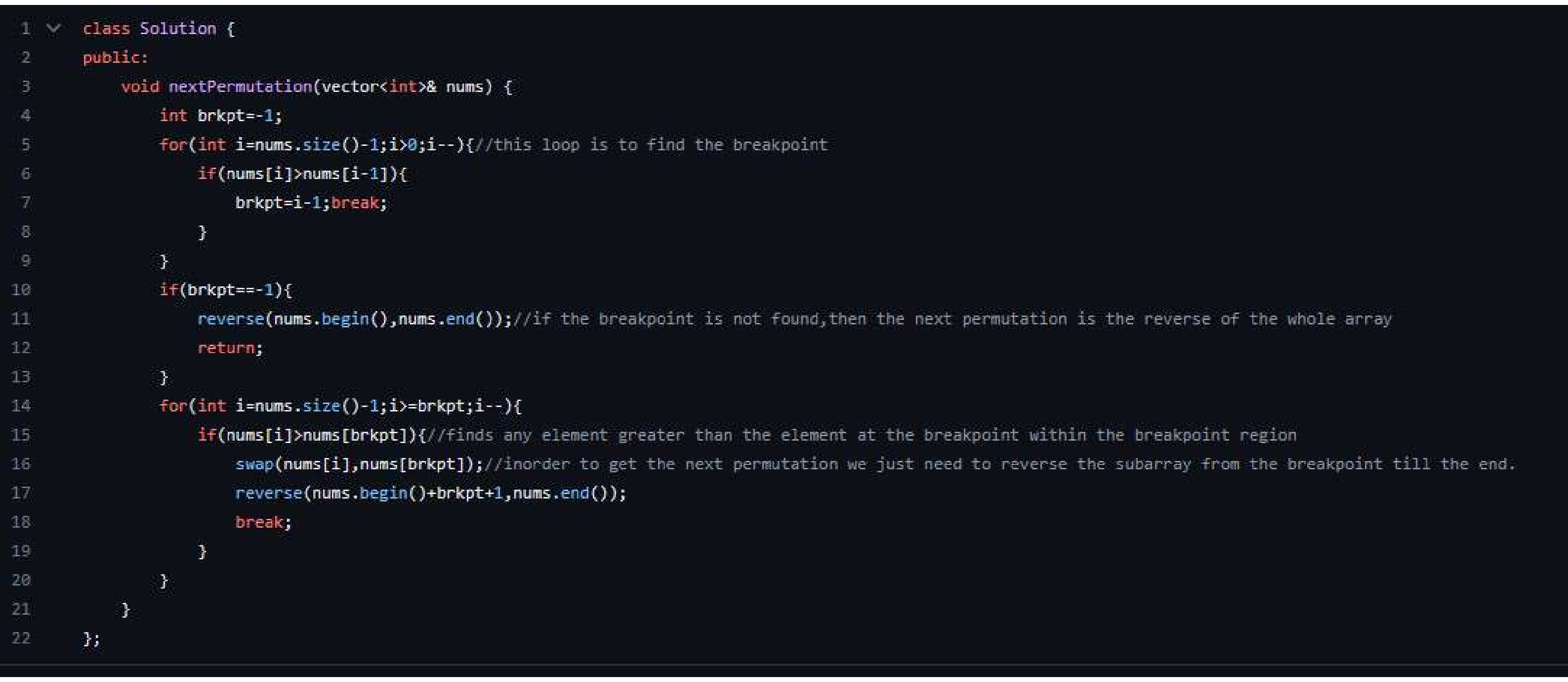
The replacement must be in place and use only constant extra memory.

Example 1:

```
Input: nums = [1,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: nums = [3,2,1]
Output: [1,2,3]
```



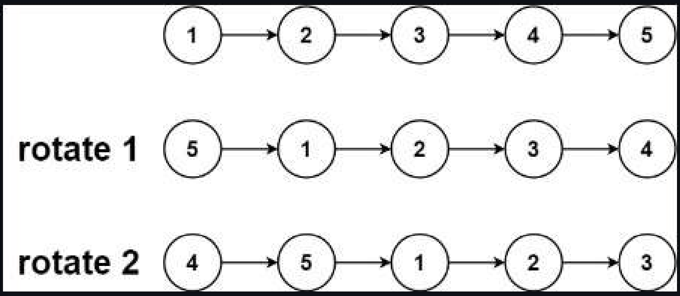
```cpp
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int brkpt=-1;
        for(int i=nums.size()-1;i>0;i--){//this loop is to find the breakpoint
            if(nums[i]>nums[i-1]){
                brkpt=i-1;break;
            }
        }
        if(brkpt==-1){
            reverse(nums.begin(),nums.end());//if the breakpoint is not found,then the next permutation is the reverse of the whole array
            return;
        }
        for(int i=nums.size()-1;i>=brkpt;i--){
            if(nums[i]>nums[brkpt]){//finds any element greater than the element at the breakpoint within the breakpoint region
                swap(nums[i],nums[brkpt]);//inorder to get the next permutation we just need to reverse the subarray from the breakpoint till the end.
                reverse(nums.begin()+brkpt+1,nums.end());
                break;
            }
        }
    }
};
```

61. Rotate List

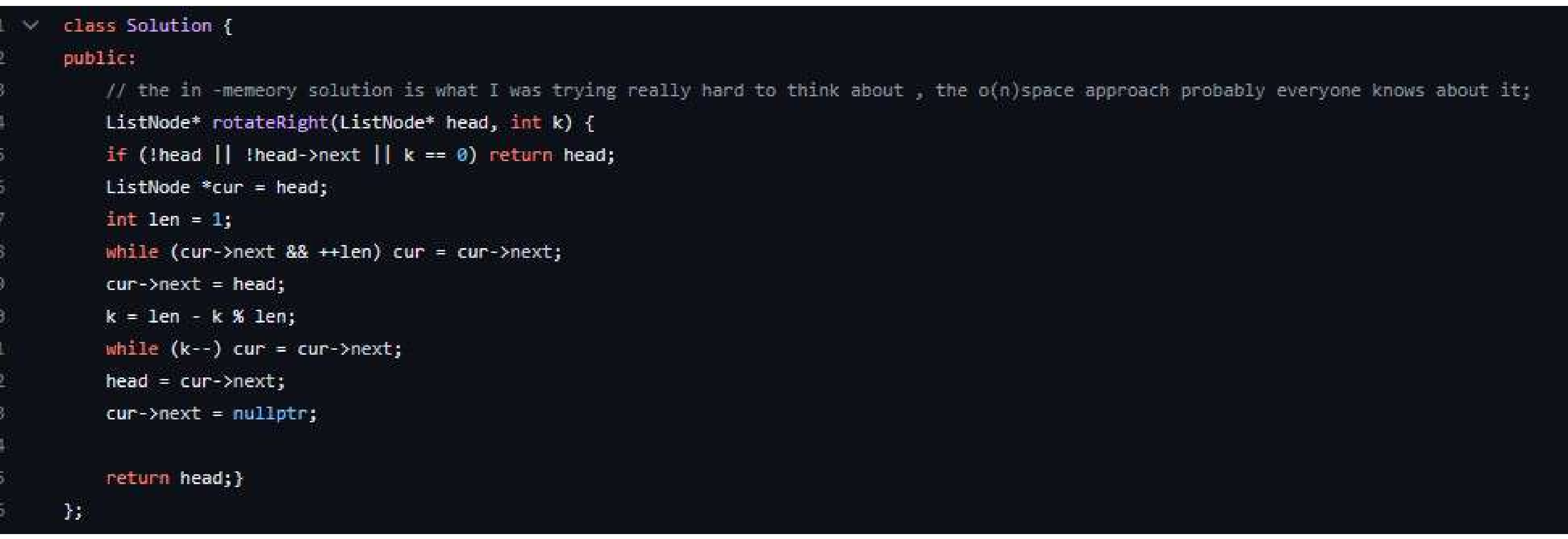
Medium

Given the `head` of a linked list, rotate the list to the right by `k` places.

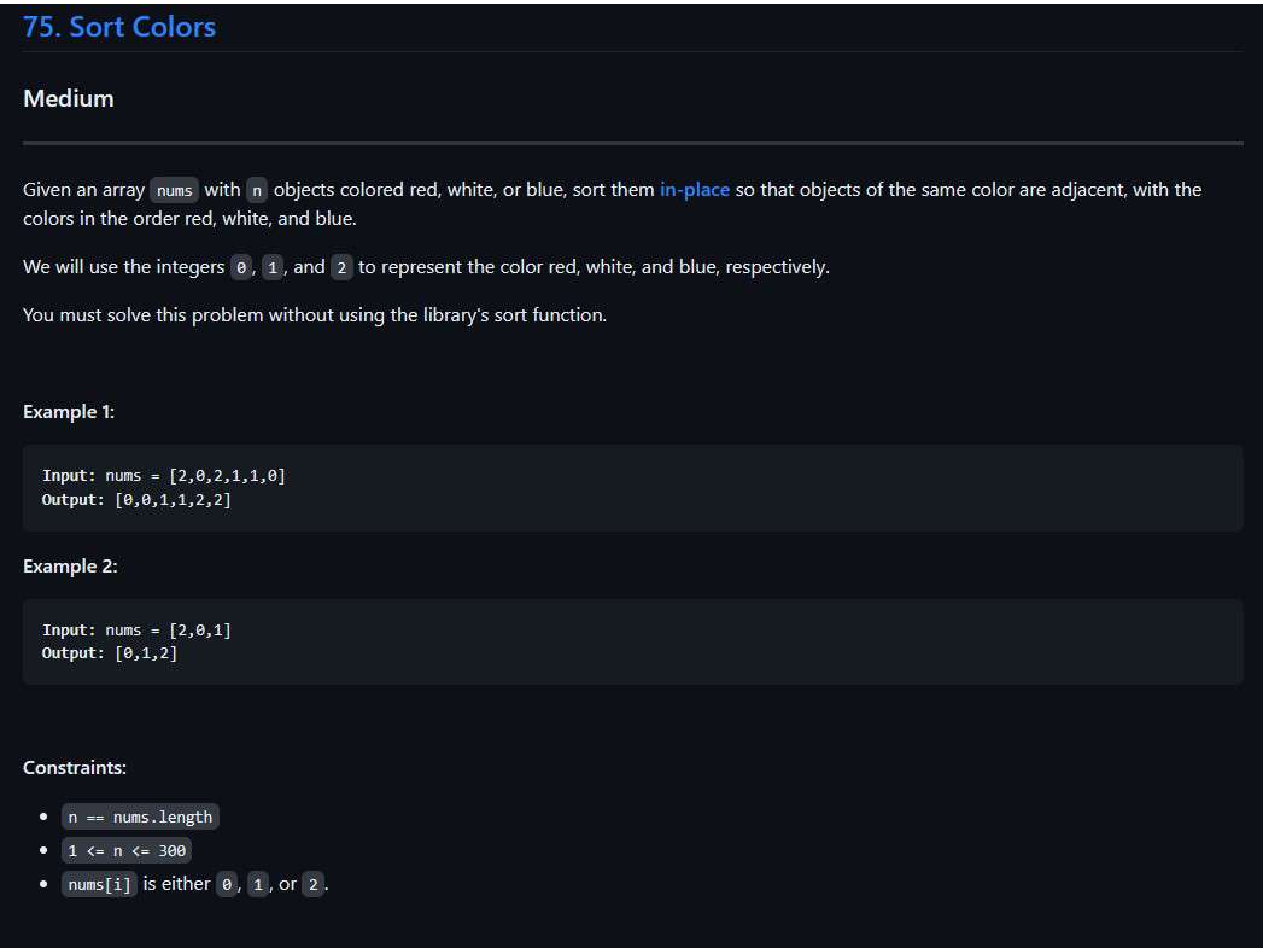
Example 1:



```
Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]
```



```
class Solution {
public:
    // the in -memeory solution is what I was trying really hard to think about , the o(n)space approach probably everyone knows about it;
    ListNode* rotateRight(ListNode* head, int k) {
    if (!head || !head->next || k == 0) return head;
    ListNode *cur = head;
    int len = 1;
    while (cur->next && ++len) cur = cur->next;
    cur->next = head;
    k = len - k % len;
    while (k--) cur = cur->next;
    head = cur->next;
    cur->next = nullptr;

    return head;}
};
```



75. Sort Colors

Medium

Given an array `nums` with `n` objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

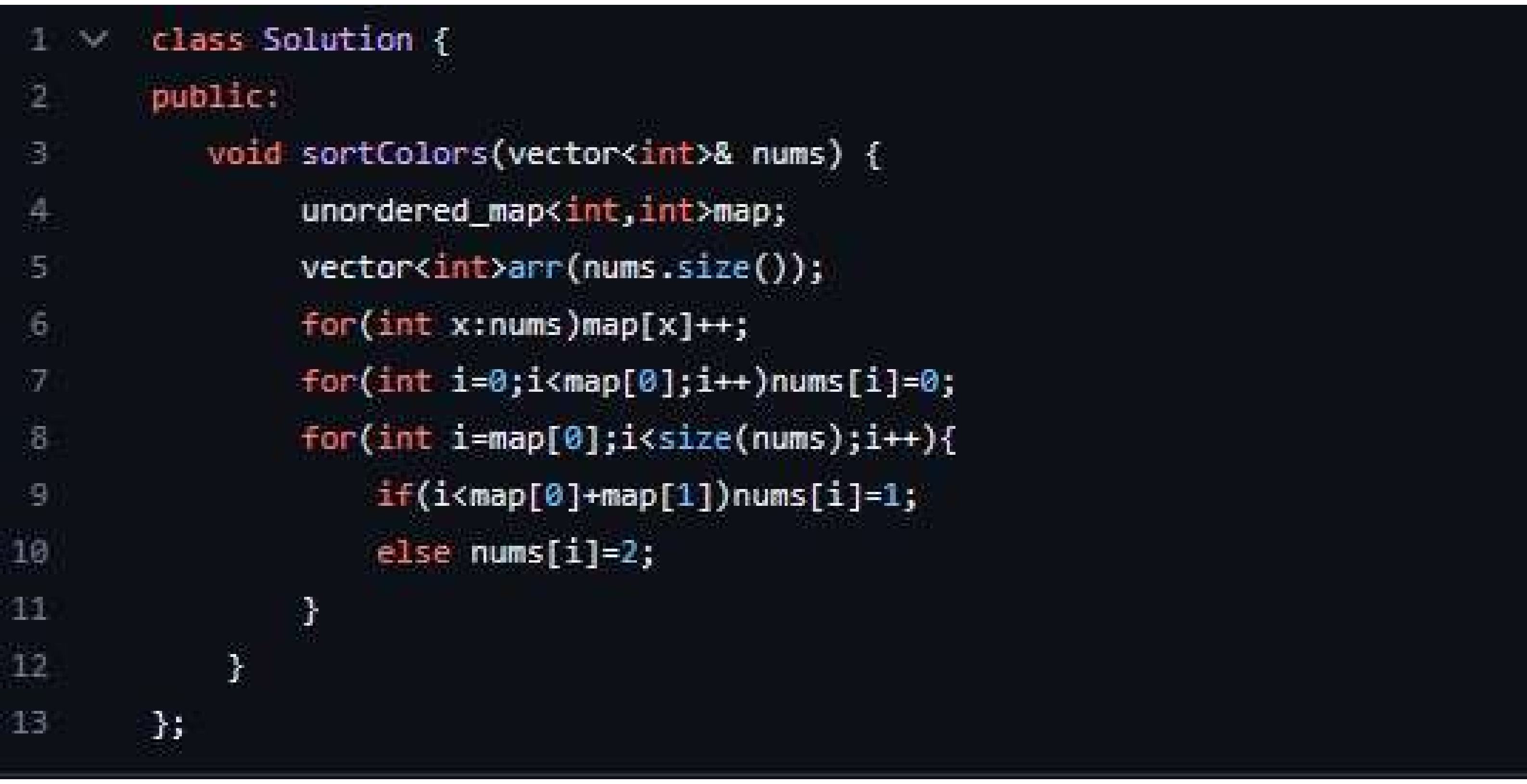
```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

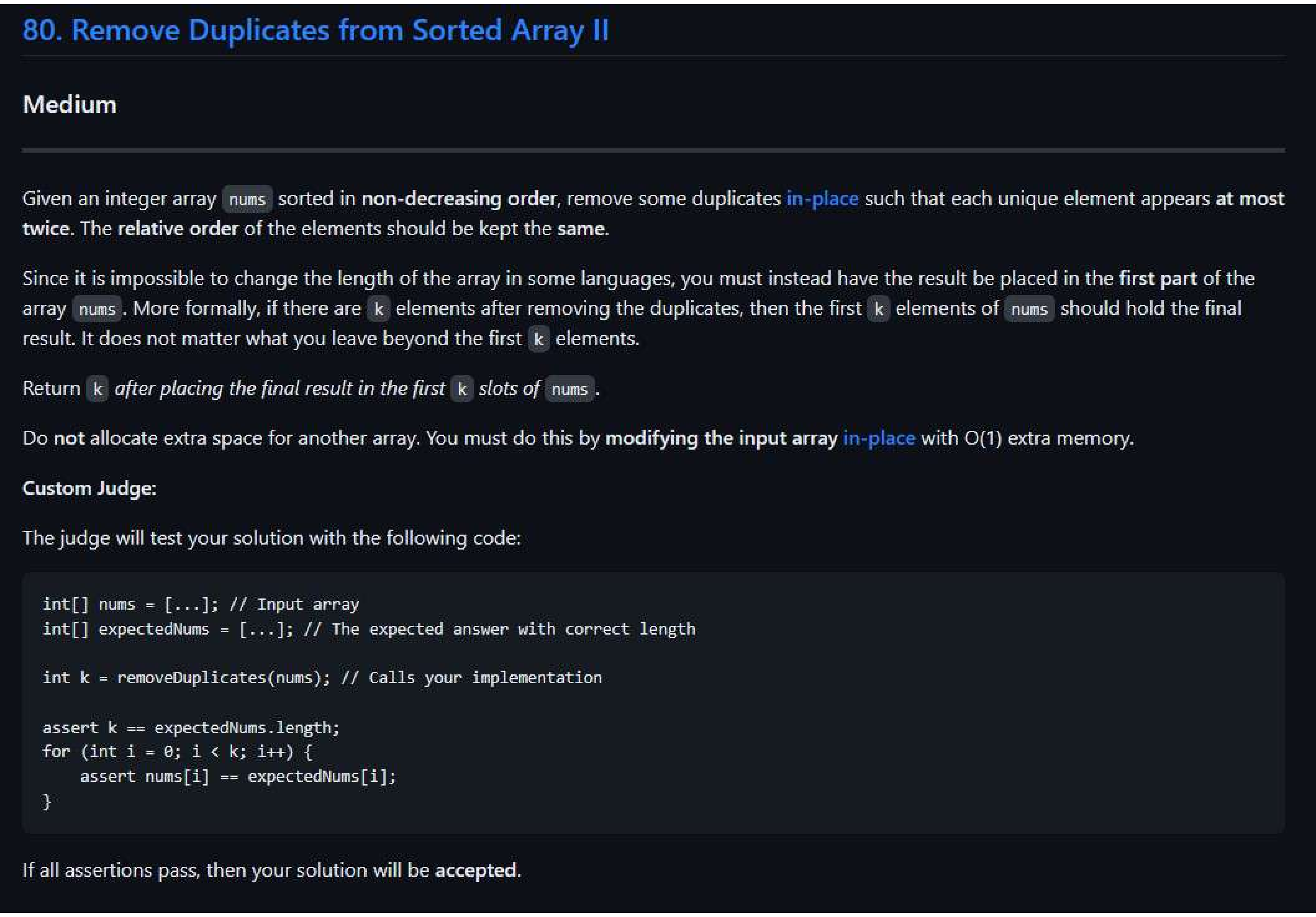
```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.



```
class Solution {
public:
    void sortColors(vector<int>& nums) {
         unordered_map<int,int>map;
         vector<int>arr(nums.size());
         for(int x:nums)map[x]++;
         for(int i=0;i<map[0];i++)nums[i]=0;
         for(int i=map[0];i<size(nums);i++){
             if(i<map[0]+map[1])nums[i]=1;
             else nums[i]=2;
         }
     }
};
```



80. Remove Duplicates from Sorted Array II

Medium

Given an integer array `nums` sorted in **non-decreasing order**, remove some duplicates in-place such that each unique element appears **at most twice.** The **relative order** of the elements should be kept the **same.**

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

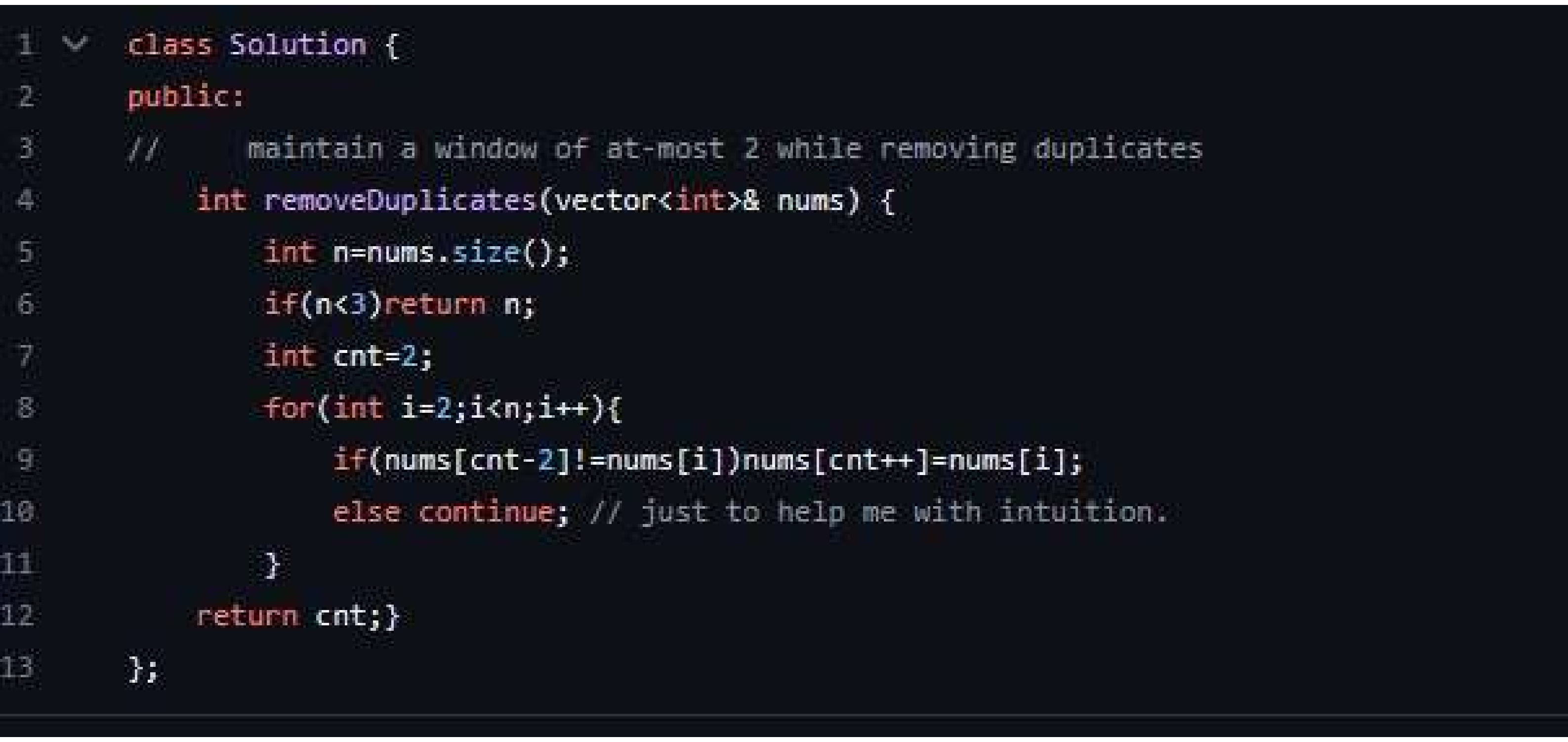
Do **not** allocate extra space for another array. You must do this by **modifying the input array** in-place with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.



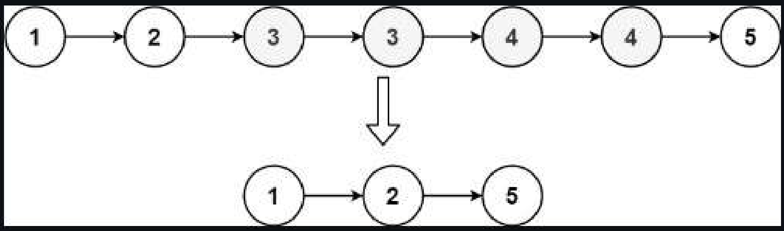
```cpp
class Solution {
public:
//     maintain a window of at-most 2 while removing duplicates
    int removeDuplicates(vector<int>& nums) {
        int n=nums.size();
        if(n<3)return n;
        int cnt=2;
        for(int i=2;i<n;i++){
            if(nums[cnt-2]!=nums[i])nums[cnt++]=nums[i];
            else continue; // just to help me with intuition.
        }
    return cnt;}
};
```

82. Remove Duplicates from Sorted List II

Medium

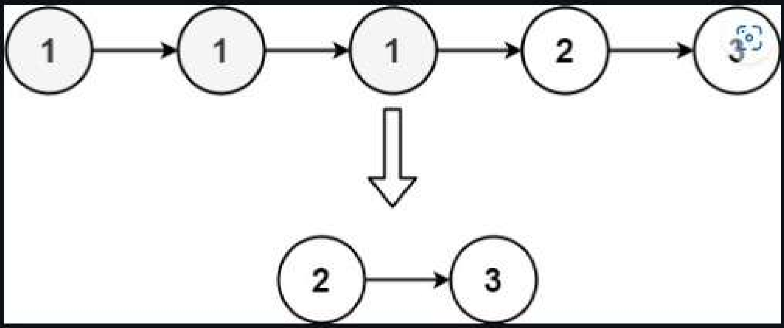
Given the `head` of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return *the linked list* ***sorted*** *as well.*

Example 1:



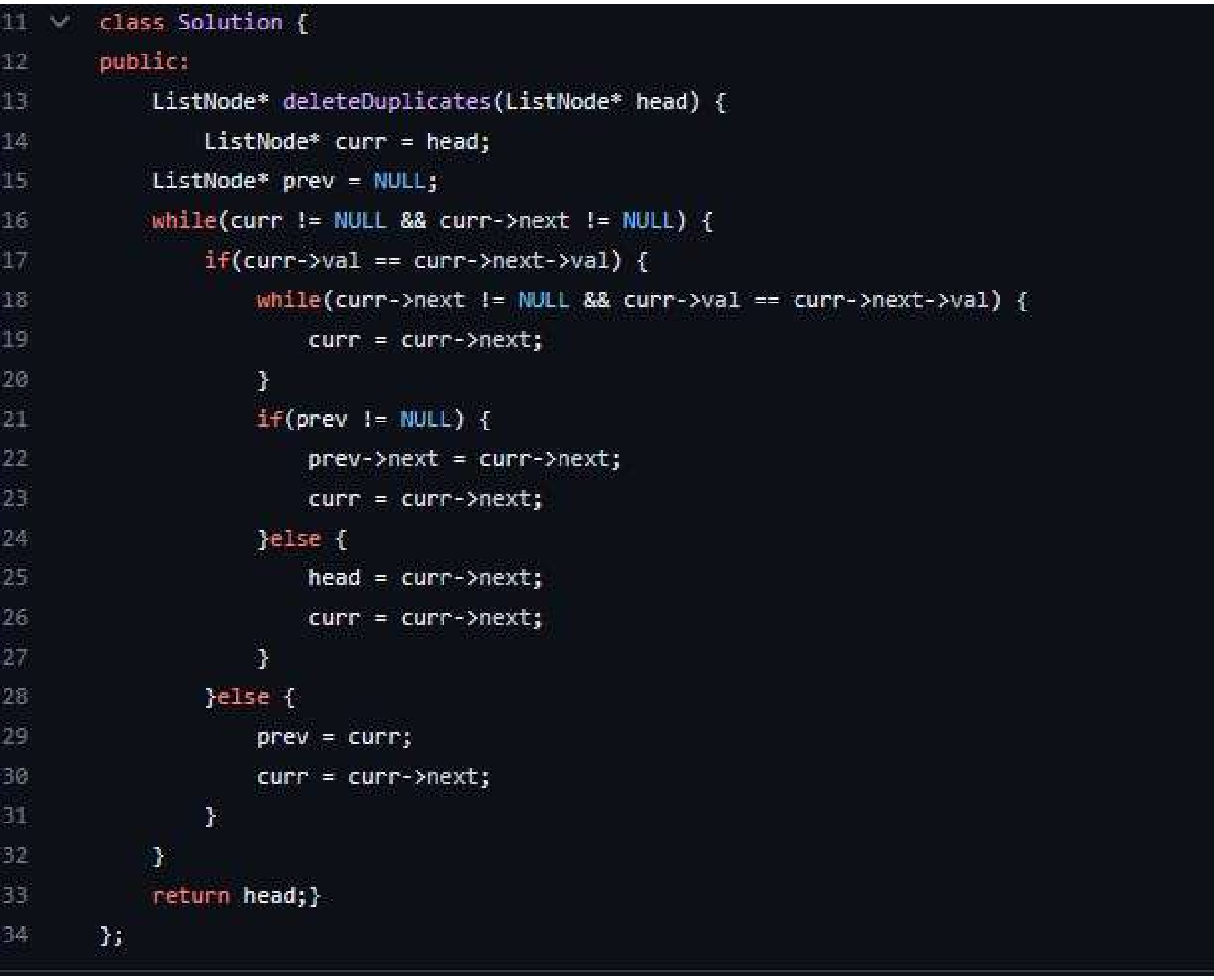
```
Input: head = [1,2,3,3,4,4,5]
Output: [1,2,5]
```

Example 2:



```
Input: head = [1,1,1,2,3]
utput: [2,3]
```

36f



```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
    ListNode* prev = NULL;
    while(curr != NULL && curr->next != NULL) {
        if(curr->val == curr->next->val) {
            while(curr->next != NULL && curr->val == curr->next->val) {
                curr = curr->next;
            }
            if(prev != NULL) {
                prev->next = curr->next;
                curr = curr->next;
            }else {
                head = curr->next;
                curr = curr->next;
            }
        }else {
            prev = curr;
            curr = curr->next;
        }
    }
    return head;}
};
```

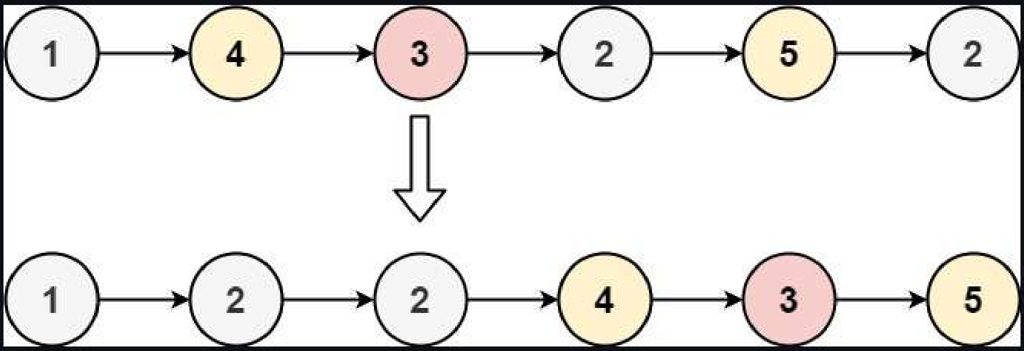
86. Partition List

Medium

Given the `head` of a linked list and a value `x`, partition it such that all nodes **less than** `x` come before nodes **greater than or equal** to `x`.

You should **preserve** the original relative order of the nodes in each of the two partitions.

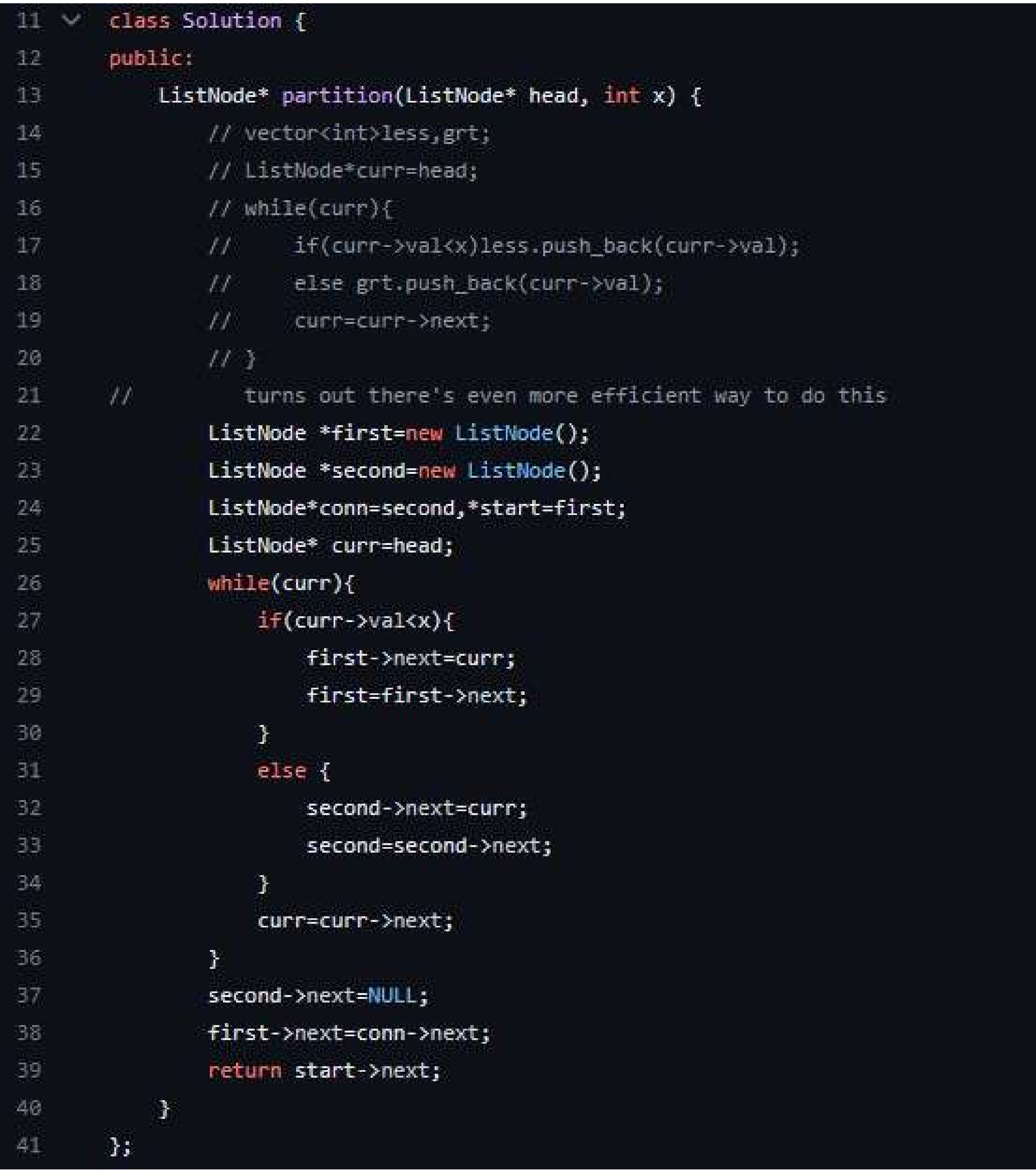
Example 1:



```
Input: head = [1,4,3,2,5,2], x = 3
Output: [1,2,2,4,3,5]
```

Example 2:

```
Input: head = [2,1], x = 2
Output: [1,2]
```



```cpp
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        // vector<int>less,grt;
        // ListNode*curr=head;
        // while(curr){
        //     if(curr->val<x)less.push_back(curr->val);
        //     else grt.push_back(curr->val);
        //     curr=curr->next;
        // }
//          turns out there's even more efficient way to do this
        ListNode *first=new ListNode();
        ListNode *second=new ListNode();
        ListNode*conn=second,*start=first;
        ListNode* curr=head;
        while(curr){
            if(curr->val<x){
                first->next=curr;
                first=first->next;
            }
            else {
                second->next=curr;
                second=second->next;
            }
            curr=curr->next;
        }
        second->next=NULL;
        first->next=conn->next;
        return start->next;
    }
};
```

142. Linked List Cycle II

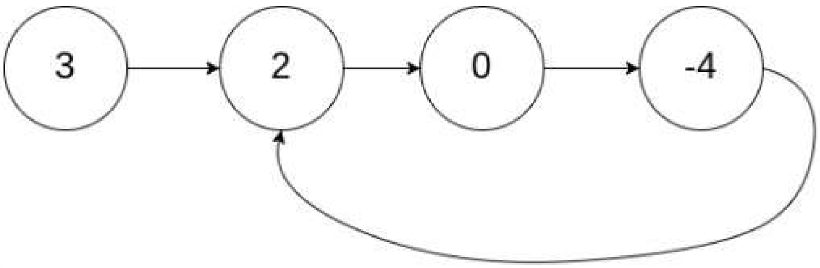
Medium 12490 884 Add to List Share

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

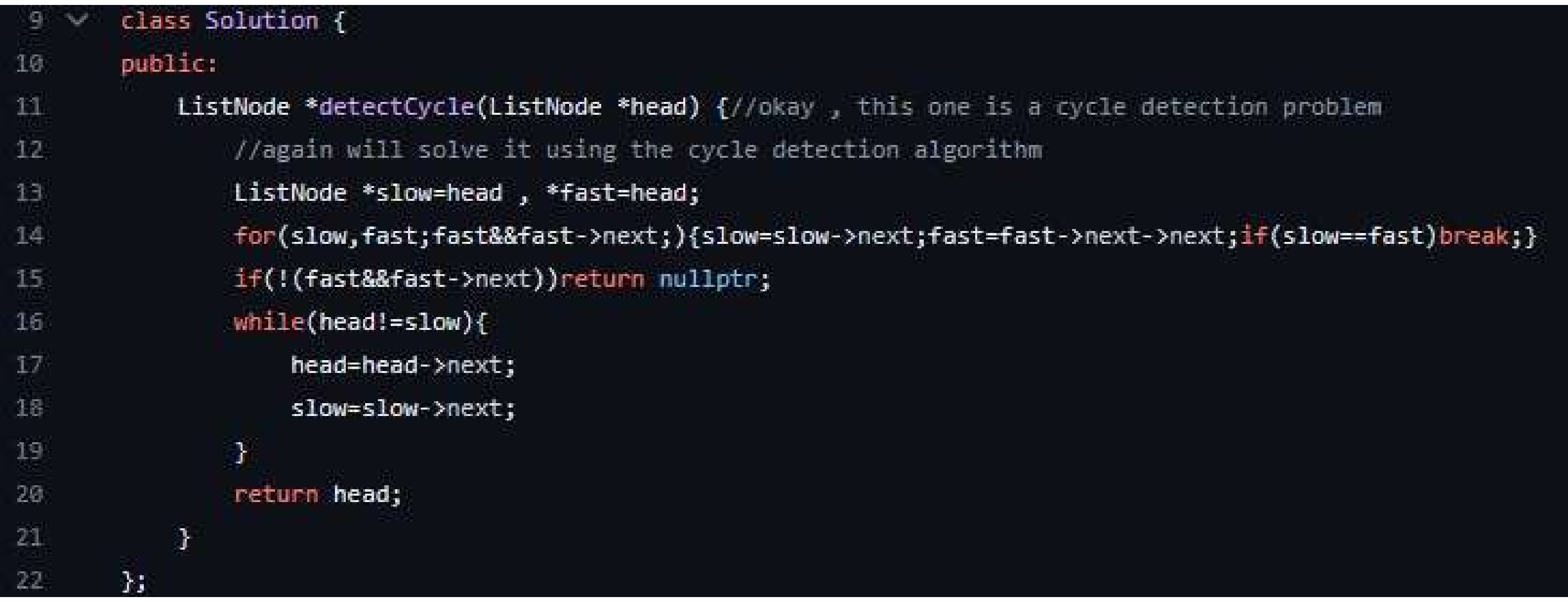
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

Example 1:



```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```



```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {//okay , this one is a cycle detection problem
        //again will solve it using the cycle detection algorithm
        ListNode *slow=head , *fast=head;
        for(slow,fast;fast&&fast->next;){slow=slow->next;fast=fast->next->next;if(slow==fast)break;}
        if(!(fast&&fast->next))return nullptr;
        while(head!=slow){
            head=head->next;
            slow=slow->next;
        }
        return head;
    }
};
```

143. Reorder List

Medium

You are given the head of a singly linked-list. The list can be represented as:

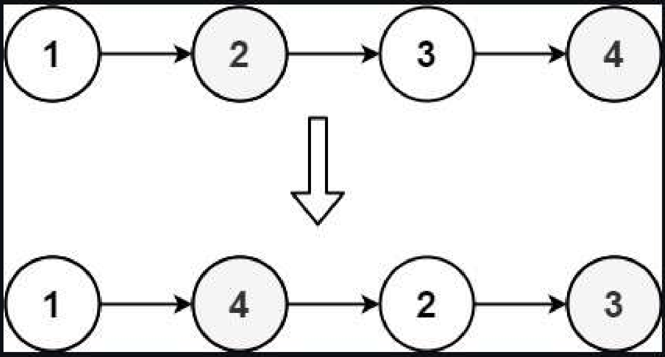
$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

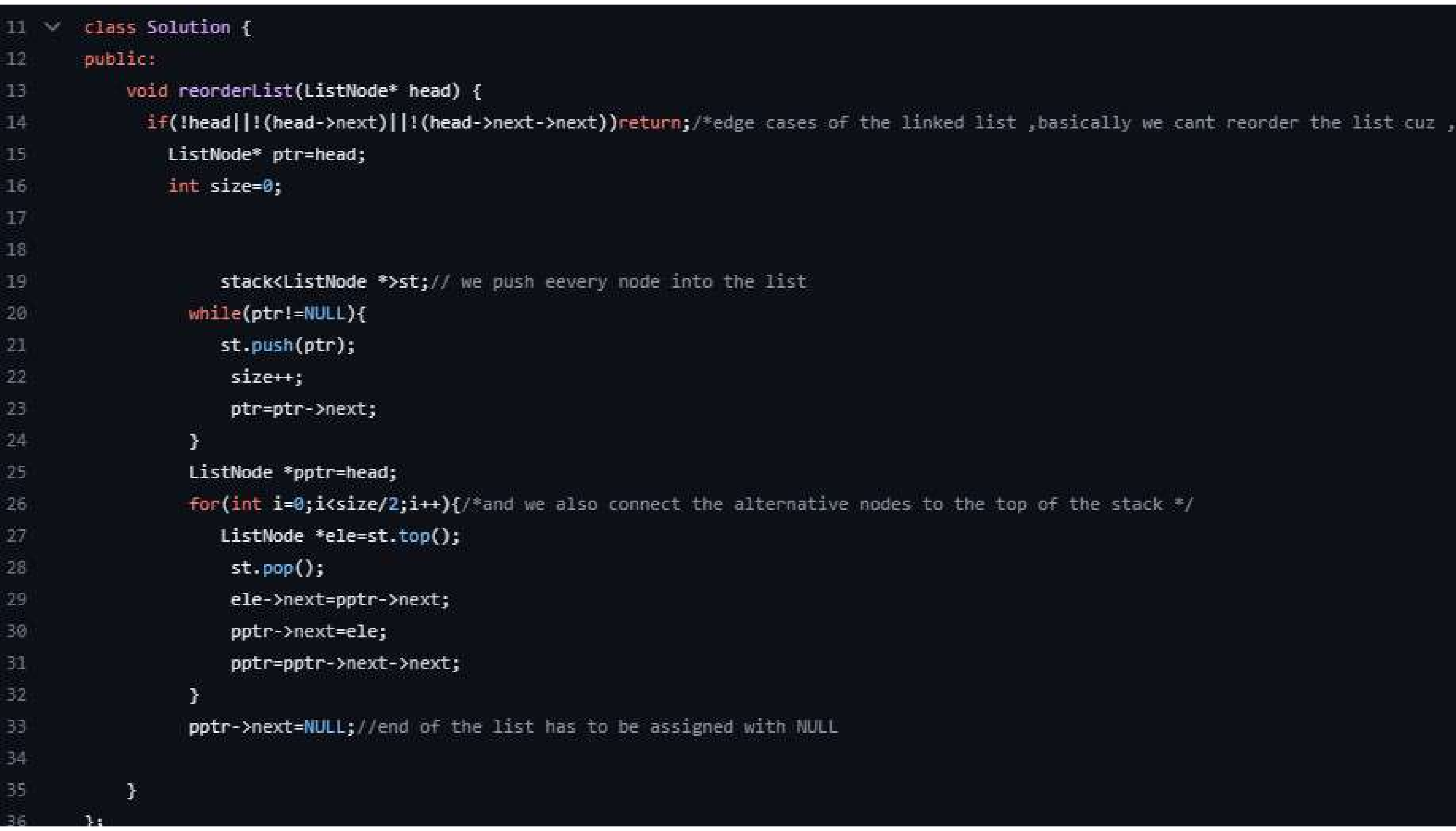
$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:



```
Input: head = [1,2,3,4]
Output: [1,4,2,3]
```



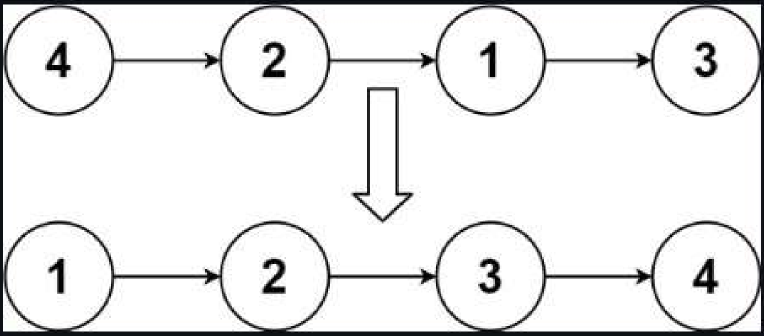
```
class Solution {
public:
    void reorderList(ListNode* head) {
      if(!head||!(head->next)||!(head->next->next))return;/*edge cases of the linked list ,basically we cant reorder the list cuz ,
        ListNode* ptr=head;
        int size=0;


             stack<ListNode *>st;// we push eevery node into the list
          while(ptr!=NULL){
             st.push(ptr);
              size++;
              ptr=ptr->next;
          }
          ListNode *pptr=head;
          for(int i=0;i<size/2;i++){/*and we also connect the alternative nodes to the top of the stack */
             ListNode *ele=st.top();
              st.pop();
              ele->next=pptr->next;
              pptr->next=ele;
              pptr=pptr->next->next;
          }
          pptr->next=NULL;//end of the list has to be assigned with NULL

    }
};
```

148. Sort List

Medium

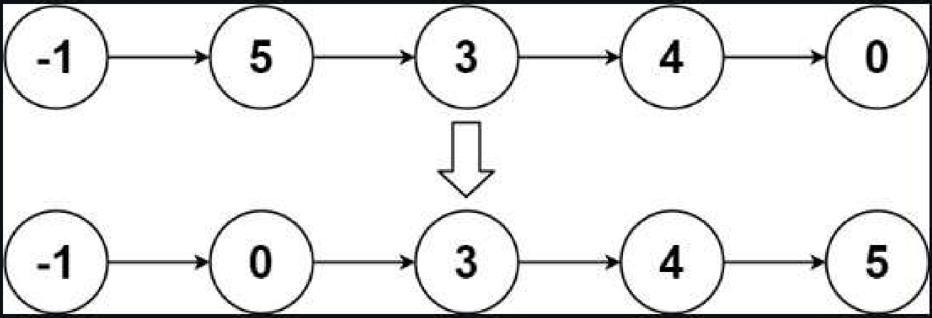
Given the `head` of a linked list, return *the list after sorting it in **ascending order***.

Example 1:

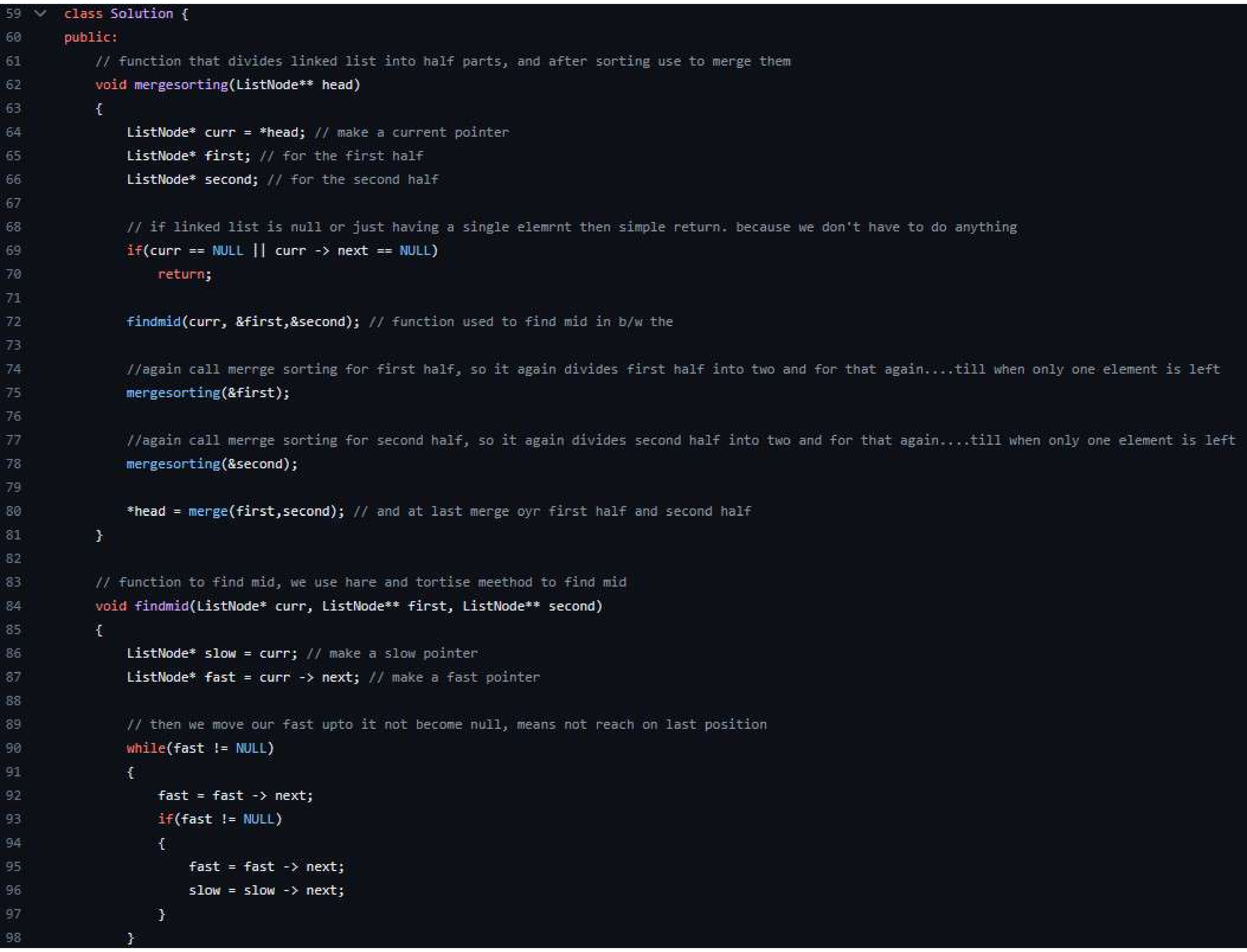


```
Input: head = [4,2,1,3]
Output: [1,2,3,4]
```

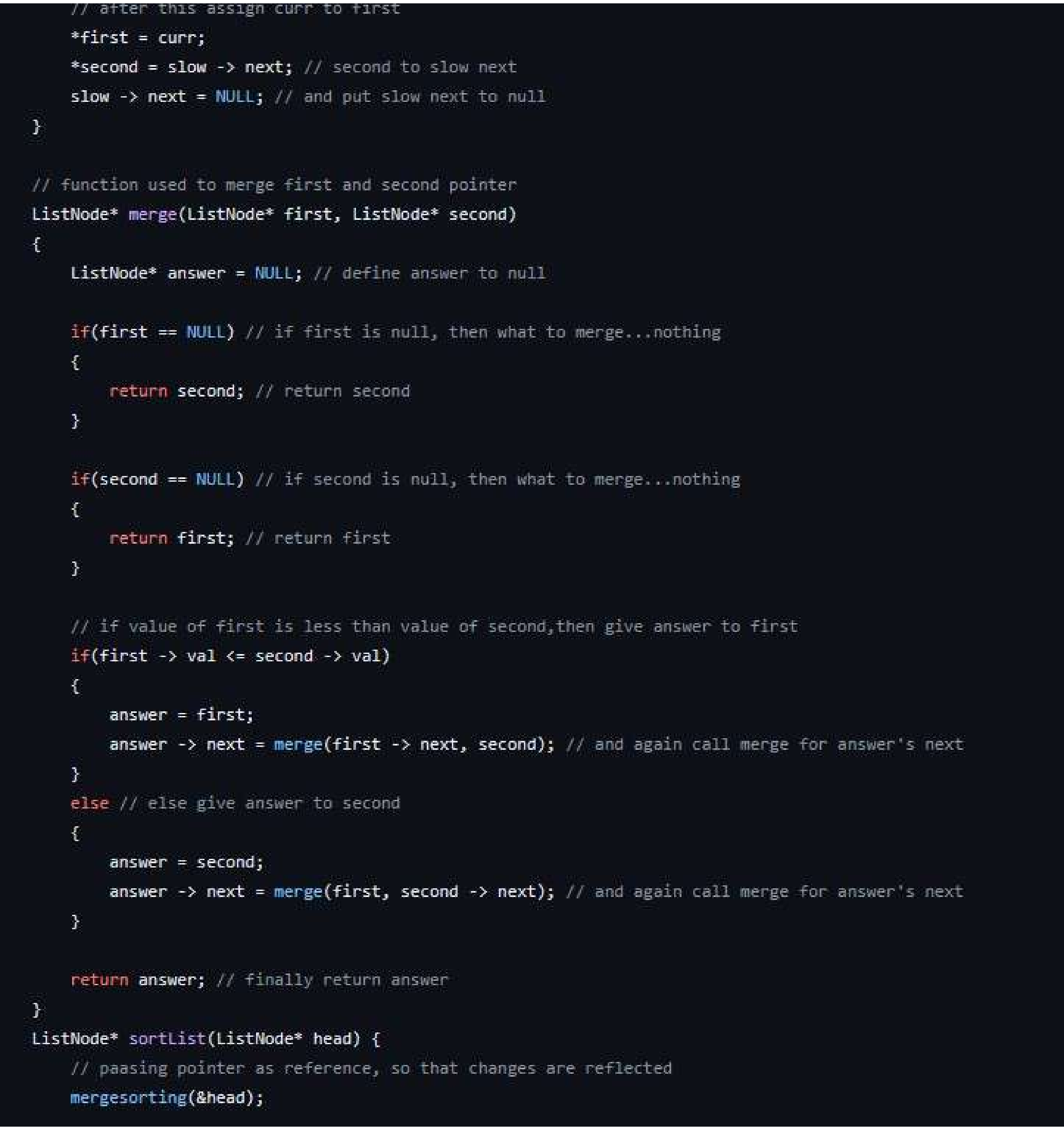
Example 2:



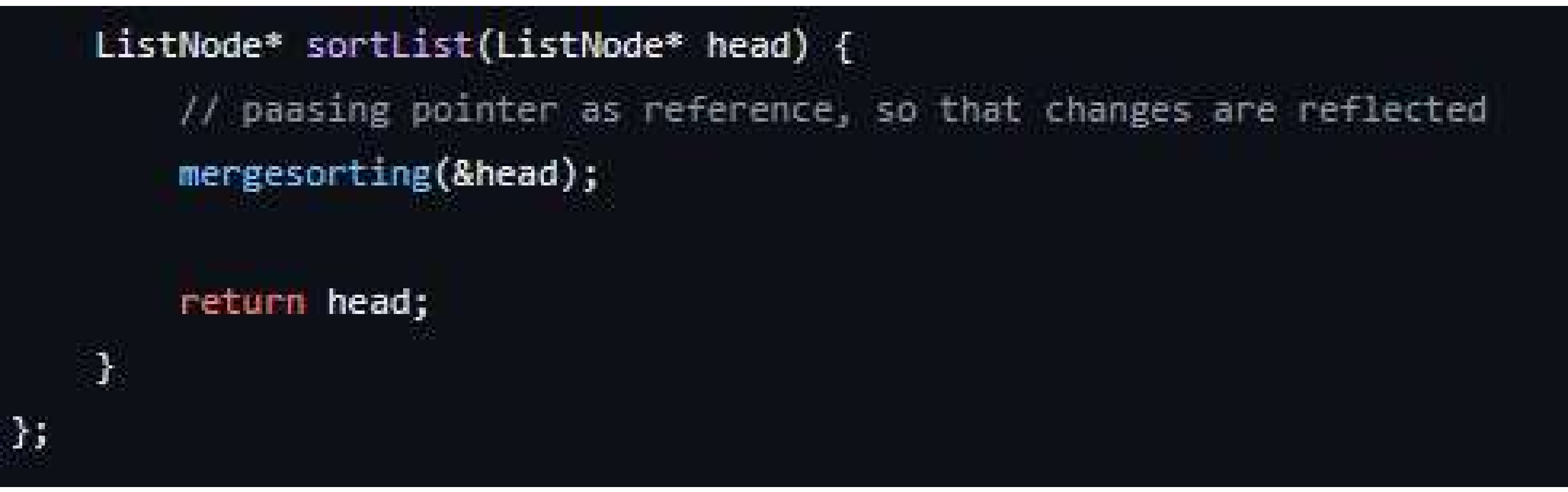
```
Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]
```



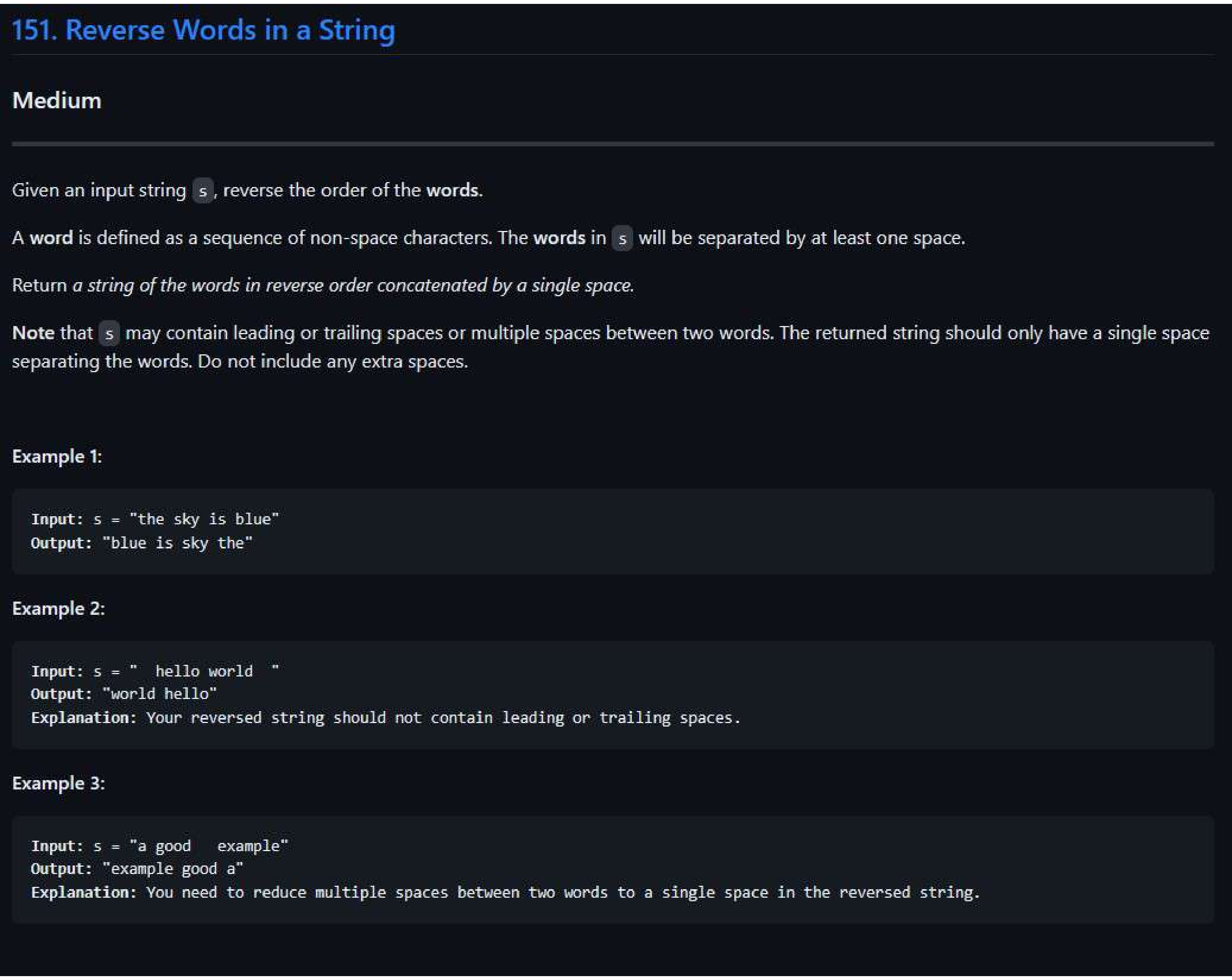
```cpp
class Solution {
public:
    // function that divides linked list into half parts, and after sorting use to merge them
    void mergesorting(ListNode** head)
    {
        ListNode* curr = *head; // make a current pointer
        ListNode* first; // for the first half
        ListNode* second; // for the second half

        // if linked list is null or just having a single elemrnt then simple return. because we don't have to do anything
        if(curr == NULL || curr -> next == NULL)
            return;

        findmid(curr, &first,&second); // function used to find mid in b/w the

        //again call merrge sorting for first half, so it again divides first half into two and for that again....till when only one element is left
        mergesorting(&first);

        //again call merrge sorting for second half, so it again divides second half into two and for that again....till when only one element is left
        mergesorting(&second);

        *head = merge(first,second); // and at last merge oyr first half and second half
    }

    // function to find mid, we use hare and tortise meethod to find mid
    void findmid(ListNode* curr, ListNode** first, ListNode** second)
    {
        ListNode* slow = curr; // make a slow pointer
        ListNode* fast = curr -> next; // make a fast pointer

        // then we move our fast upto it not become null, means not reach on last position
        while(fast != NULL)
        {
            fast = fast -> next;
            if(fast != NULL)
            {
                fast = fast -> next;
                slow = slow -> next;
            }
        }
```



```
    // after this assign curr to first
    *first = curr;
    *second = slow -> next; // second to slow next
    slow -> next = NULL; // and put slow next to null
}

// function used to merge first and second pointer
ListNode* merge(ListNode* first, ListNode* second)
{
    ListNode* answer = NULL; // define answer to null

    if(first == NULL) // if first is null, then what to merge...nothing
    {
        return second; // return second
    }

    if(second == NULL) // if second is null, then what to merge...nothing
    {
        return first; // return first
    }

    // if value of first is less than value of second,then give answer to first
    if(first -> val <= second -> val)
    {
        answer = first;
        answer -> next = merge(first -> next, second); // and again call merge for answer's next
    }
    else // else give answer to second
    {
        answer = second;
        answer -> next = merge(first, second -> next); // and again call merge for answer's next
    }

    return answer; // finally return answer
}
ListNode* sortList(ListNode* head) {
    // paasing pointer as reference, so that changes are reflected
    mergesorting(&head);
```



```
    ListNode* sortList(ListNode* head) {
        // paasing pointer as reference, so that changes are reflected
        mergesorting(&head);

        return head;
    }
};
```



151. Reverse Words in a String

Medium

Given an input string `s`, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in `s` will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

Note that `s` may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

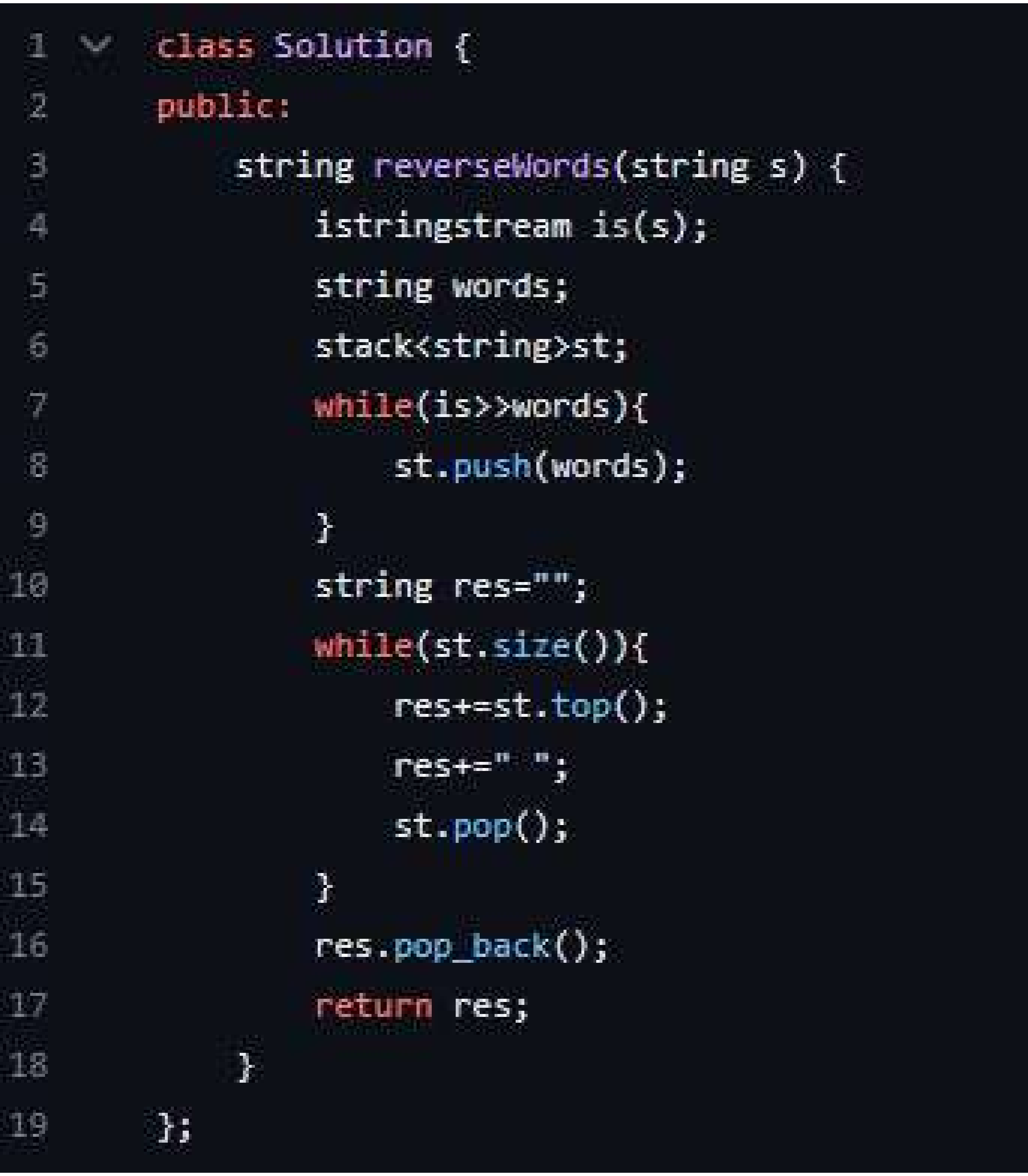
```
Input: s = "the sky is blue"
Output: "blue is sky the"
```

Example 2:

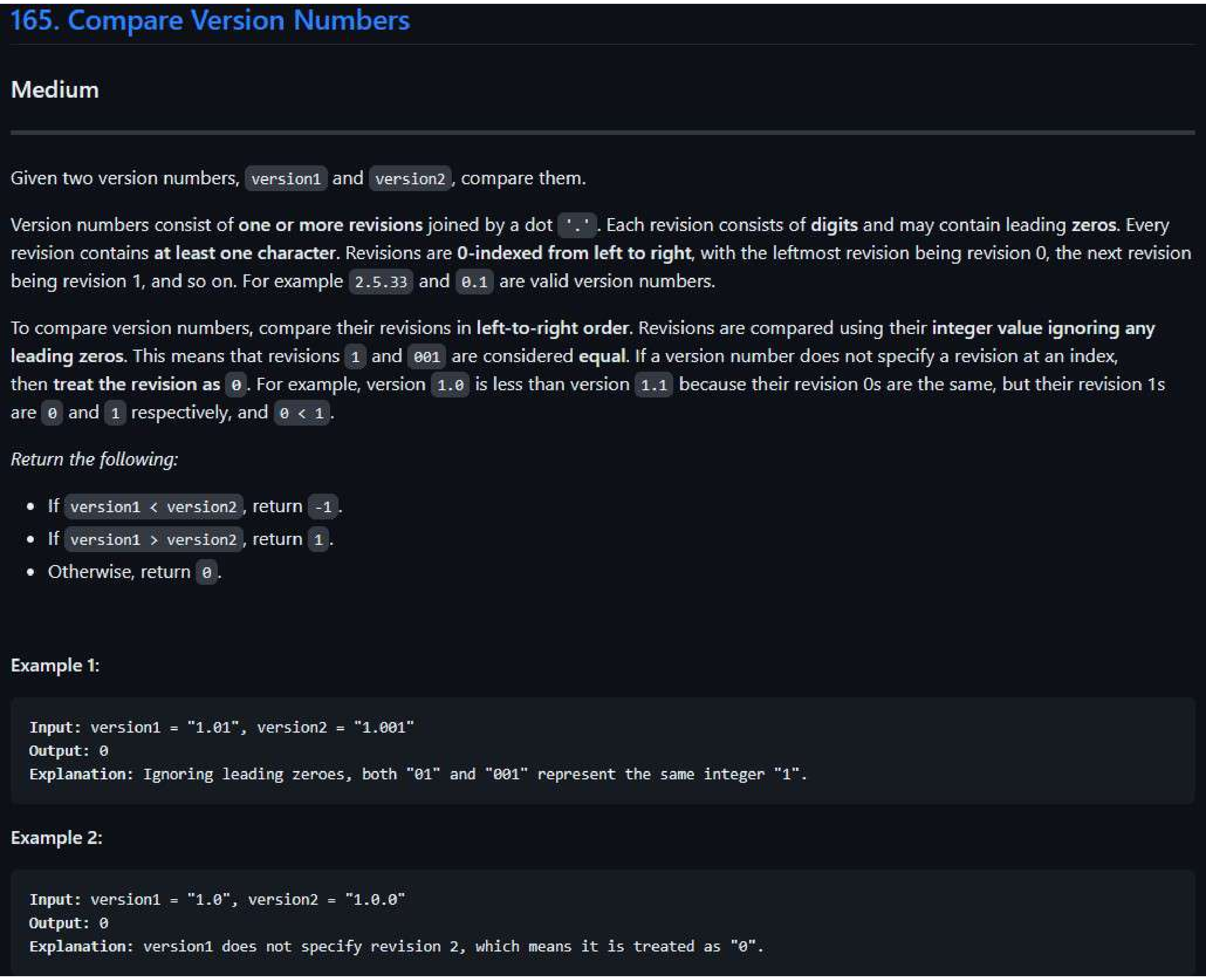
```
Input: s = "  hello world  "
Output: "world hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: s = "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string.
```



```cpp
class Solution {
public:
    string reverseWords(string s) {
        istringstream is(s);
        string words;
        stack<string>st;
        while(is>>words){
            st.push(words);
        }
        string res="";
        while(st.size()){
            res+=st.top();
            res+=" ";
            st.pop();
        }
        res.pop_back();
        return res;
    }
};
```



165. Compare Version Numbers

Medium

Given two version numbers, `version1` and `version2`, compare them.

Version numbers consist of **one or more revisions** joined by a dot `'.'`. Each revision consists of **digits** and may contain leading **zeros**. Every revision contains **at least one character**. Revisions are **0-indexed from left to right**, with the leftmost revision being revision 0, the next revision being revision 1, and so on. For example `2.5.33` and `0.1` are valid version numbers.

To compare version numbers, compare their revisions in **left-to-right order**. Revisions are compared using their **integer value ignoring any leading zeros**. This means that revisions `1` and `001` are considered **equal**. If a version number does not specify a revision at an index, then **treat the revision as** `0`. For example, version `1.0` is less than version `1.1` because their revision 0s are the same, but their revision 1s are `0` and `1` respectively, and `0 < 1`.

Return the following:

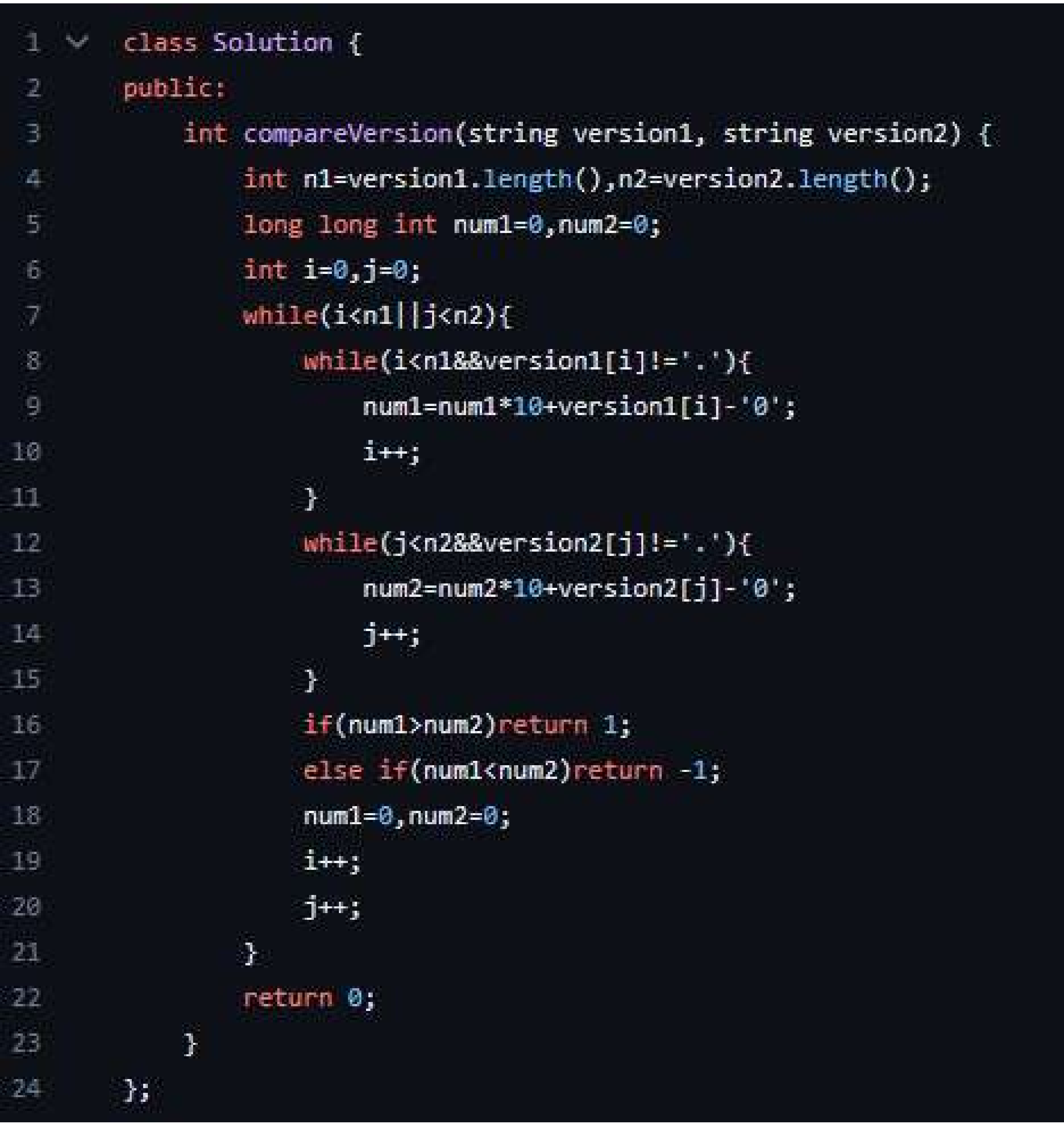
- If `version1 < version2`, return `-1`.
- If `version1 > version2`, return `1`.
- Otherwise, return `0`.

Example 1:

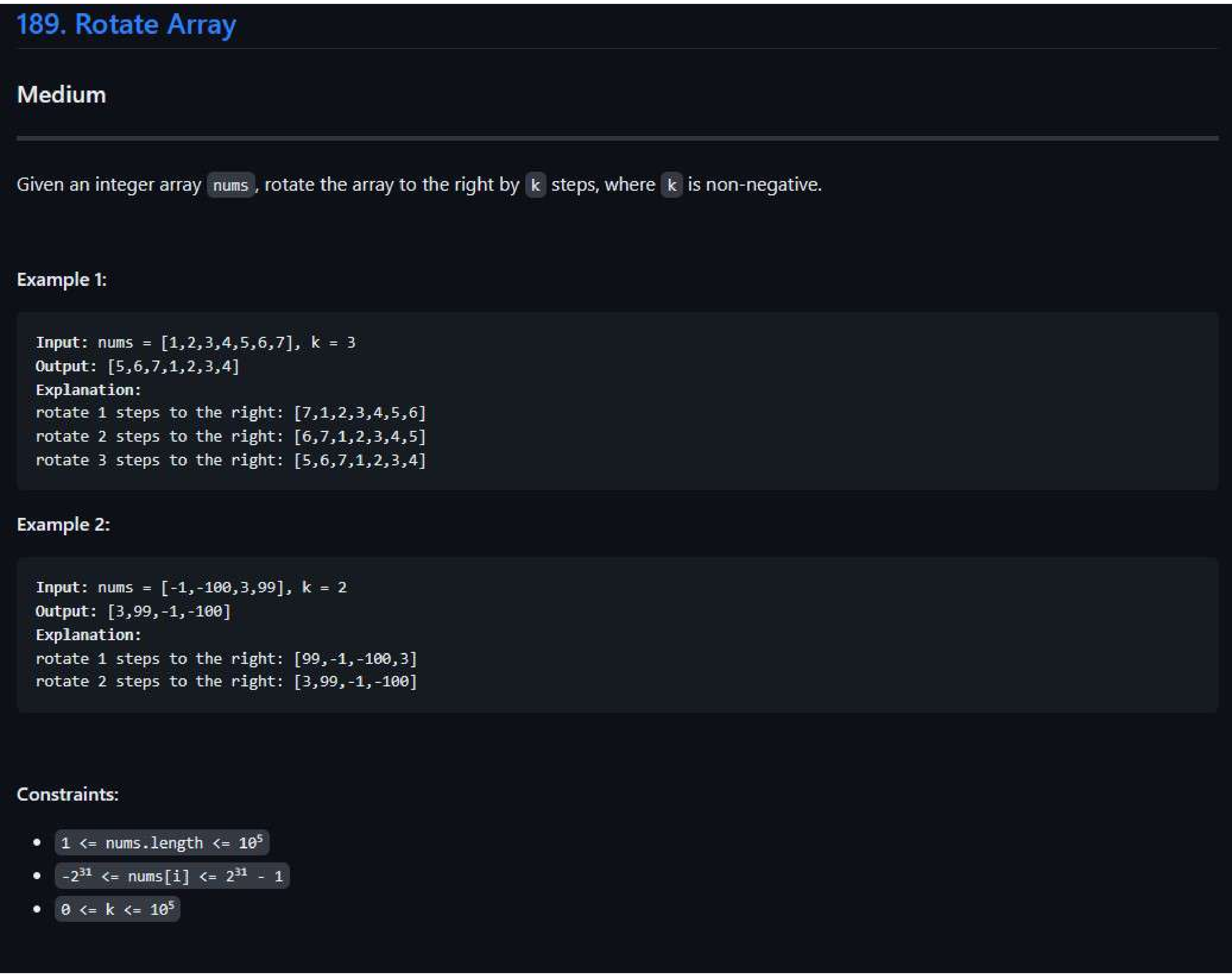
```
Input: version1 = "1.01", version2 = "1.001"
Output: 0
Explanation: Ignoring leading zeroes, both "01" and "001" represent the same integer "1".
```

Example 2:

```
Input: version1 = "1.0", version2 = "1.0.0"
Output: 0
Explanation: version1 does not specify revision 2, which means it is treated as "0".
```



```cpp
class Solution {
public:
    int compareVersion(string version1, string version2) {
        int n1=version1.length(),n2=version2.length();
        long long int num1=0,num2=0;
        int i=0,j=0;
        while(i<n1||j<n2){
            while(i<n1&&version1[i]!='.'){
                num1=num1*10+version1[i]-'0';
                i++;
            }
            while(j<n2&&version2[j]!='.'){
                num2=num2*10+version2[j]-'0';
                j++;
            }
            if(num1>num2)return 1;
            else if(num1<num2)return -1;
            num1=0,num2=0;
            i++;
            j++;
        }
        return 0;
    }
};
```



189. Rotate Array

Medium

Given an integer array `nums`, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:

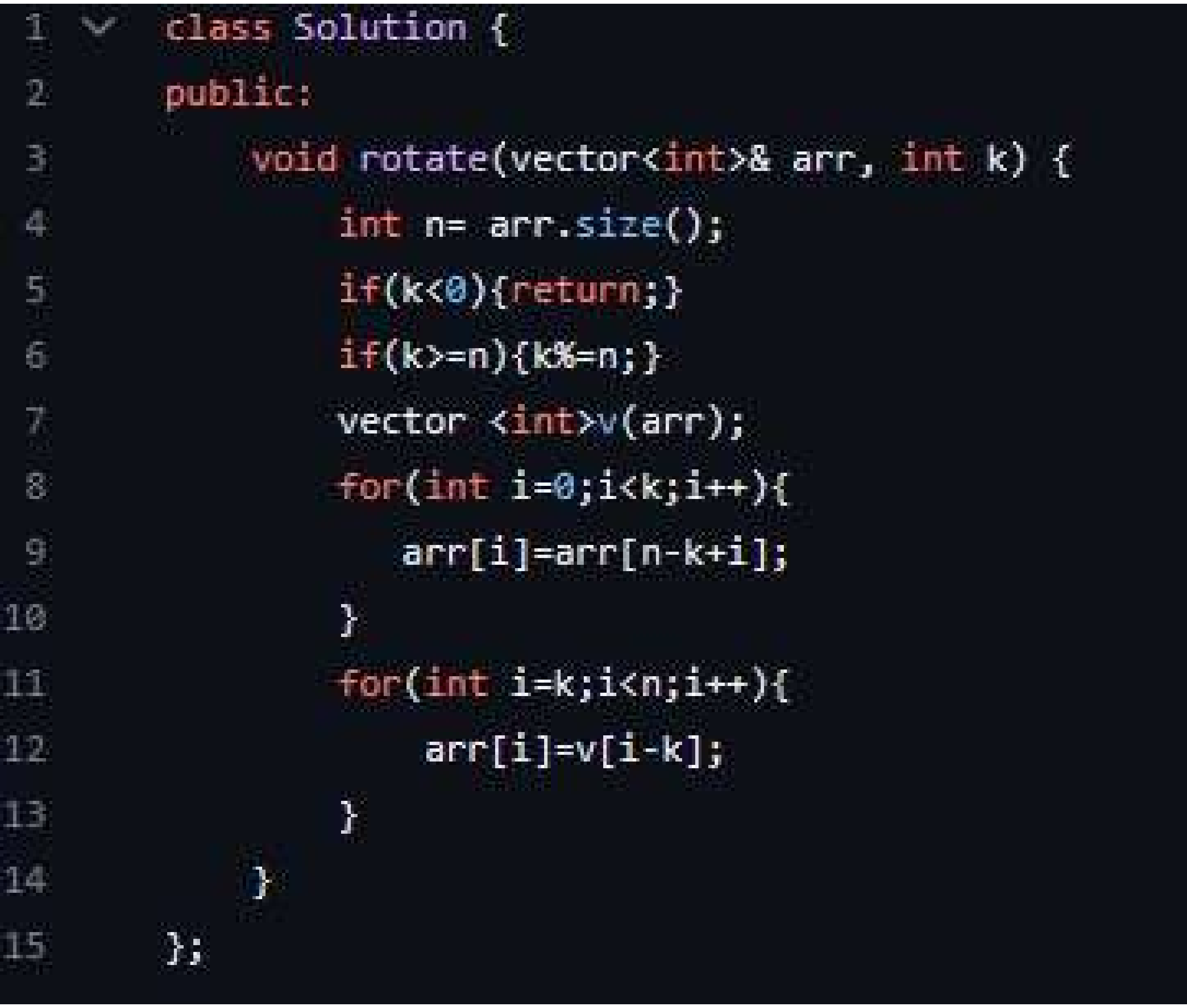
```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

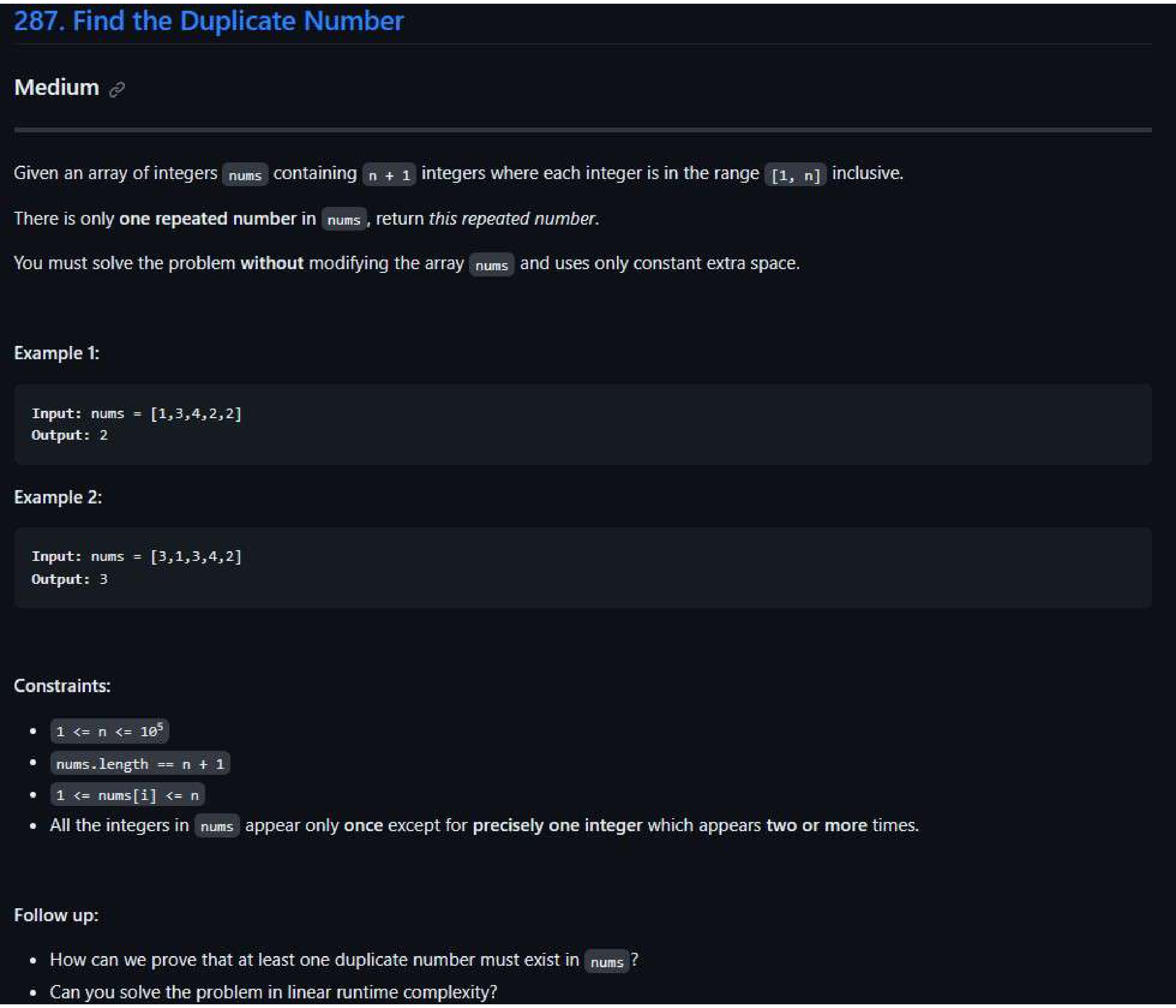
```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Constraints:

- $1 <= nums.length <= 10^5$
- $-2^{31} <= nums[i] <= 2^{31} - 1$
- $0 <= k <= 10^5$



```cpp
class Solution {
public:
    void rotate(vector<int>& arr, int k) {
        int n= arr.size();
        if(k<0){return;}
        if(k>=n){k%=n;}
        vector <int>v(arr);
        for(int i=0;i<k;i++){
           arr[i]=arr[n-k+i];
        }
        for(int i=k;i<n;i++){
            arr[i]=v[i-k];
        }
    }
};
```



287. Find the Duplicate Number

Medium

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and uses only constant extra space.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

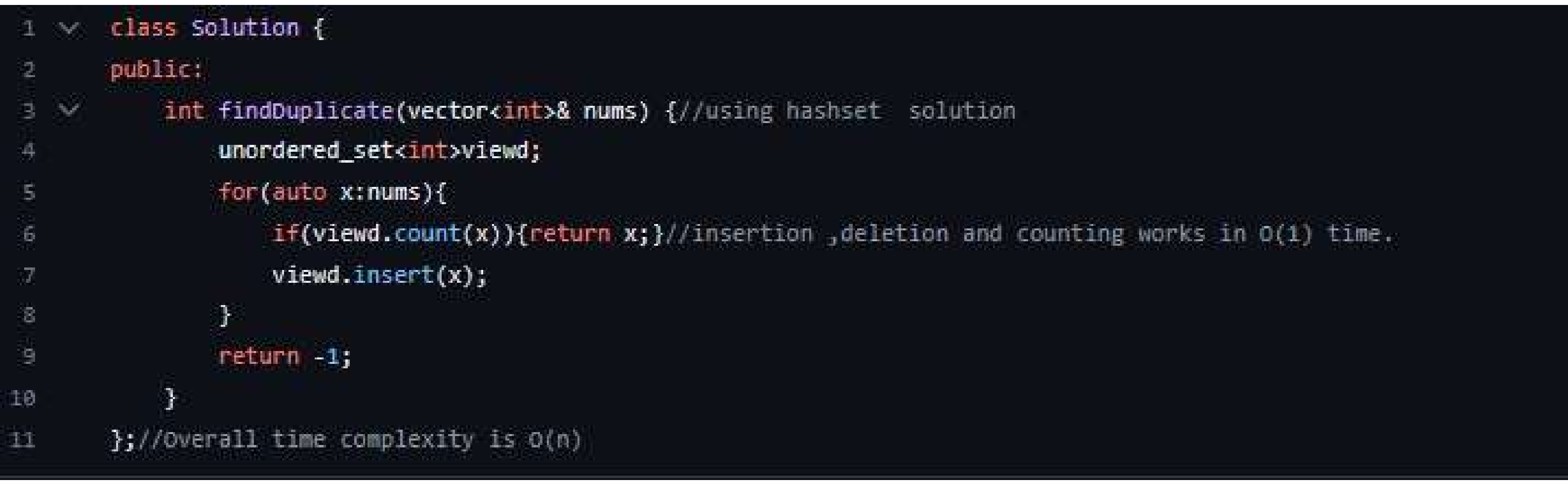
```
Input: nums = [3,1,3,4,2]
Output: 3
```

Constraints:

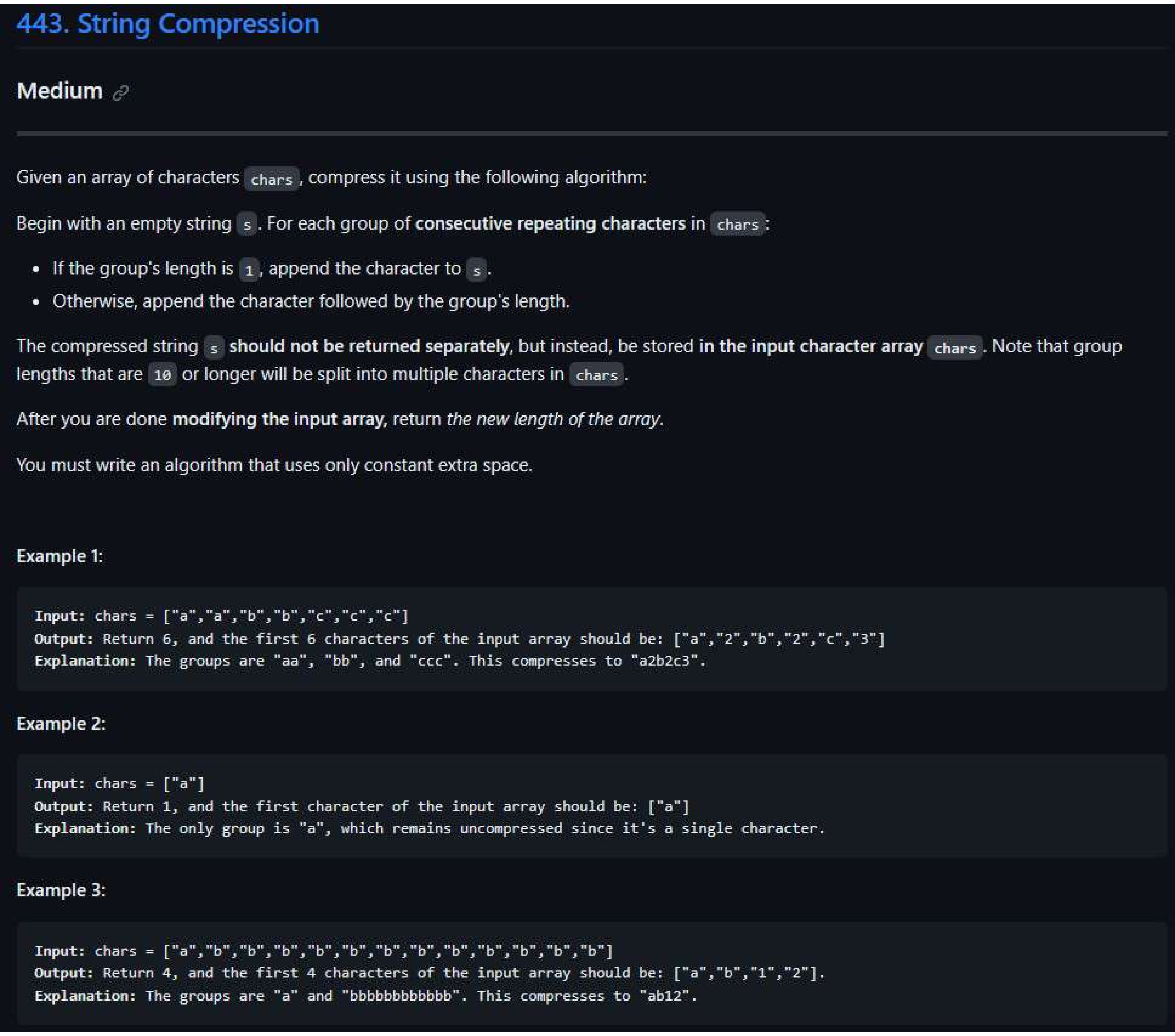
- $1 <= n <= 10^5$
- `nums.length == n + 1`
- `1 <= nums[i] <= n`
- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

Follow up:

- How can we prove that at least one duplicate number must exist in `nums`?
- Can you solve the problem in linear runtime complexity?



```cpp
class Solution {
public:
    int findDuplicate(vector<int>& nums) {//using hashset  solution
        unordered_set<int>viewd;
        for(auto x:nums){
            if(viewd.count(x)){return x;}//insertion ,deletion and counting works in O(1) time.
            viewd.insert(x);
        }
        return -1;
    }
};//Overall time complexity is O(n)
```



443. String Compression

Medium

Given an array of characters `chars`, compress it using the following algorithm:

Begin with an empty string `s`. For each group of **consecutive repeating characters** in `chars`:

- If the group's length is `1`, append the character to `s`.
- Otherwise, append the character followed by the group's length.

The compressed string `s` **should not be returned separately,** but instead, be stored **in the input character array** `chars`. Note that group lengths that are `10` or longer will be split into multiple characters in `chars`.

After you are done **modifying the input array,** return *the new length of the array*.

You must write an algorithm that uses only constant extra space.

Example 1:

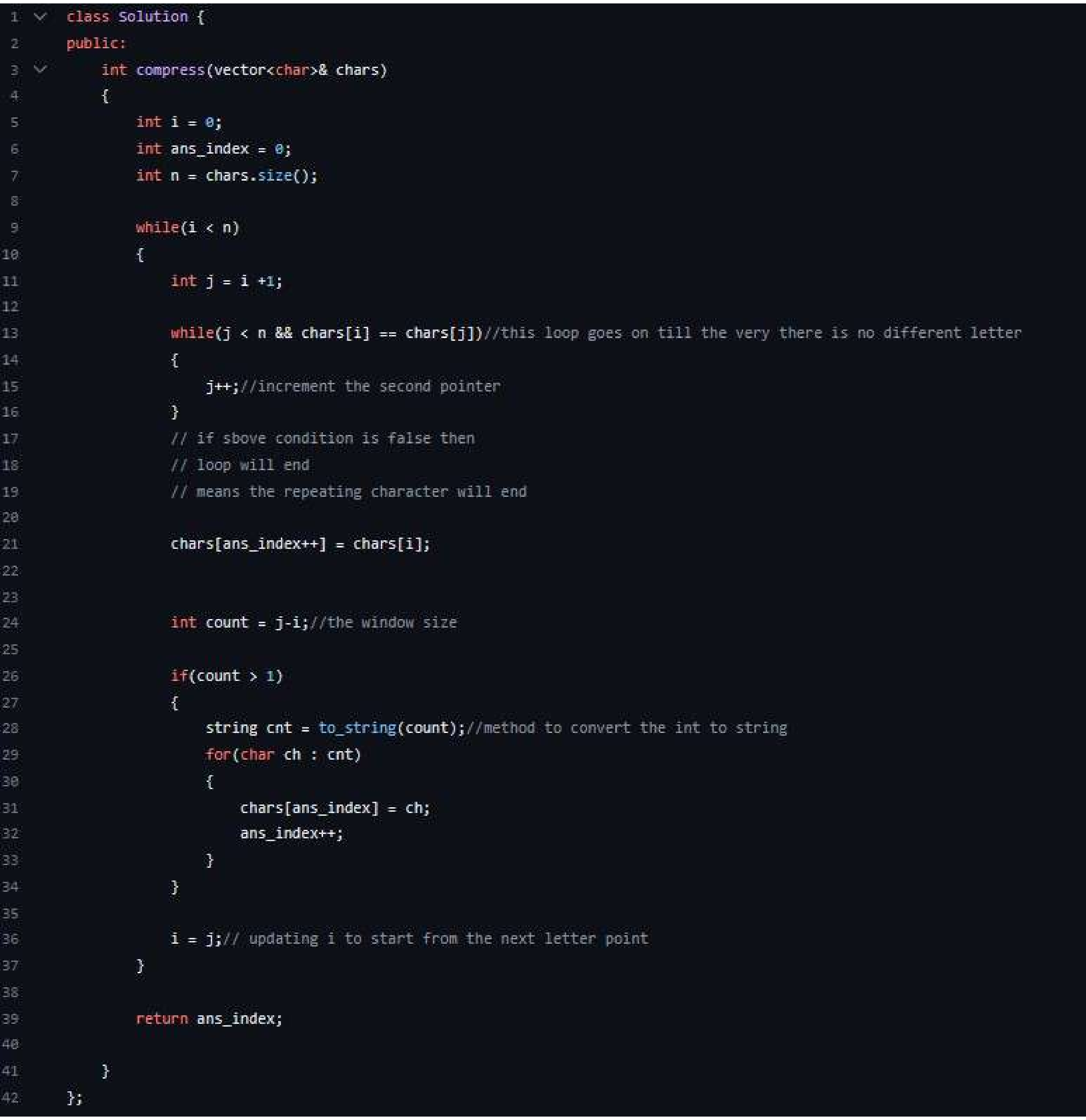
```
Input: chars = ["a","a","b","b","c","c","c"]
Output: Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]
Explanation: The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".
```

Example 2:

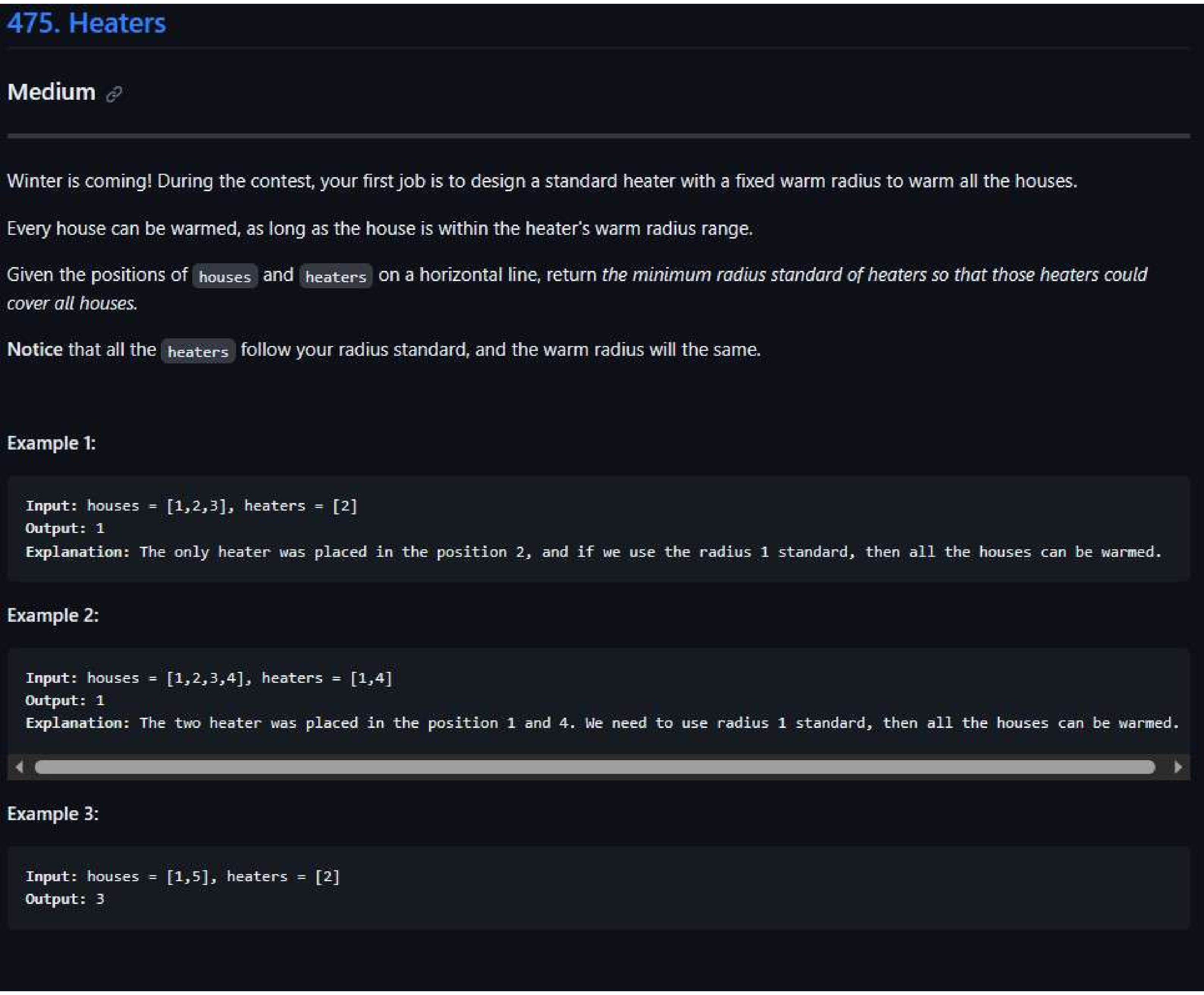
```
Input: chars = ["a"]
Output: Return 1, and the first character of the input array should be: ["a"]
Explanation: The only group is "a", which remains uncompressed since it's a single character.
```

Example 3:

```
Input: chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]
Output: Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].
Explanation: The groups are "a" and "bbbbbbbbbbbb". This compresses to "ab12".
```



```cpp
class Solution {
public:
    int compress(vector<char>& chars)
    {
        int i = 0;
        int ans_index = 0;
        int n = chars.size();

        while(i < n)
        {
            int j = i +1;

            while(j < n && chars[i] == chars[j])//this loop goes on till the very there is no different letter
            {
                j++;//increment the second pointer
            }
            // if sbove condition is false then
            // loop will end
            // means the repeating character will end

            chars[ans_index++] = chars[i];


            int count = j-i;//the window size

            if(count > 1)
            {
                string cnt = to_string(count);//method to convert the int to string
                for(char ch : cnt)
                {
                    chars[ans_index] = ch;
                    ans_index++;
                }
            }

            i = j;// updating i to start from the next letter point
        }

        return ans_index;

    }
};
```



475. Heaters

Medium

Winter is coming! During the contest, your first job is to design a standard heater with a fixed warm radius to warm all the houses.

Every house can be warmed, as long as the house is within the heater's warm radius range.

Given the positions of `houses` and `heaters` on a horizontal line, return *the minimum radius standard of heaters so that those heaters could cover all houses.*

Notice that all the `heaters` follow your radius standard, and the warm radius will the same.

Example 1:

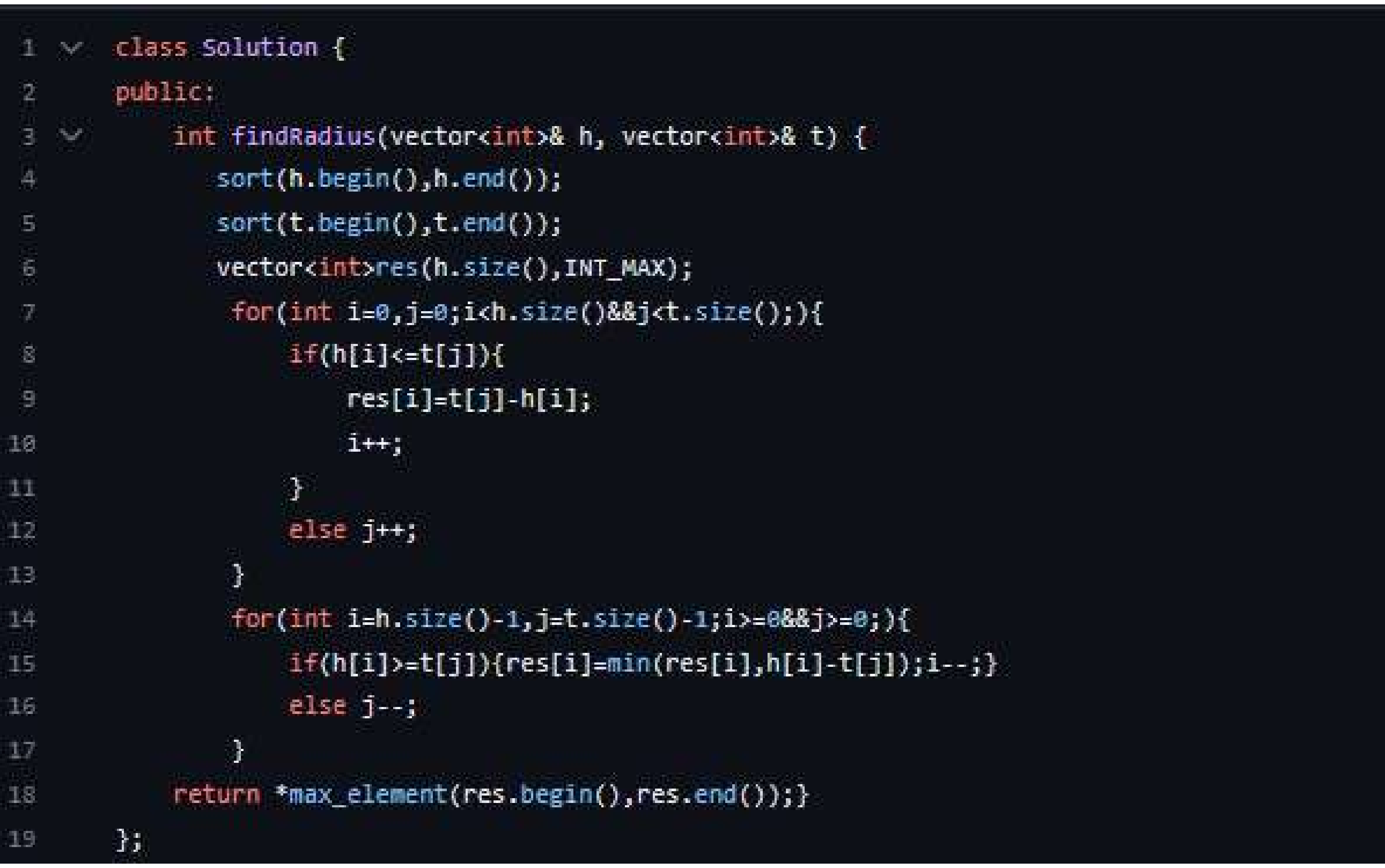
```
Input: houses = [1,2,3], heaters = [2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.
```

Example 2:

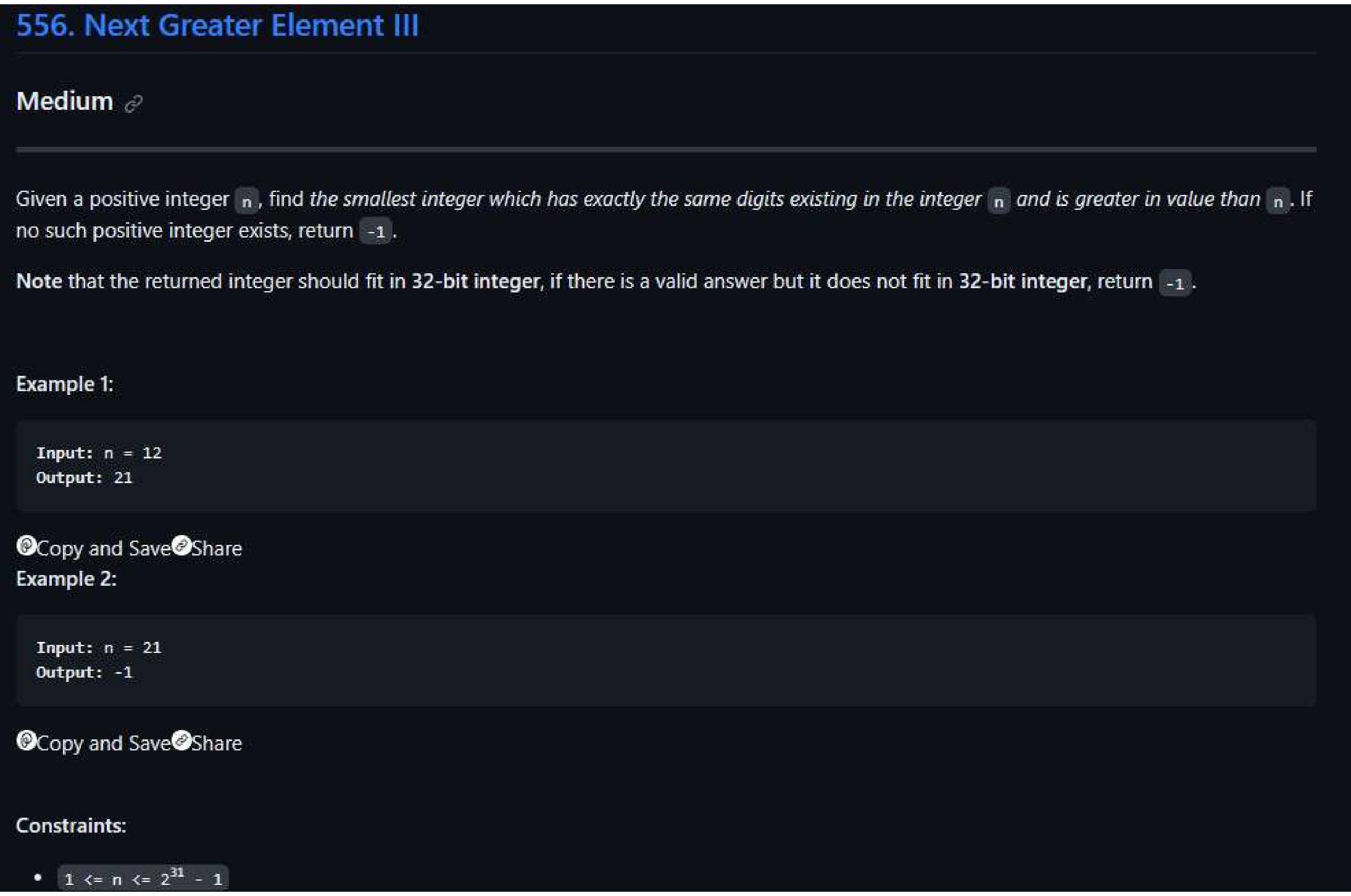
```
Input: houses = [1,2,3,4], heaters = [1,4]
Output: 1
Explanation: The two heater was placed in the position 1 and 4. We need to use radius 1 standard, then all the houses can be warmed.
```

Example 3:

```
Input: houses = [1,5], heaters = [2]
Output: 3
```



```
class Solution {
public:
    int findRadius(vector<int>& h, vector<int>& t) {
       sort(h.begin(),h.end());
       sort(t.begin(),t.end());
       vector<int>res(h.size(),INT_MAX);
        for(int i=0,j=0;i<h.size()&&j<t.size();){
            if(h[i]<=t[j]){
                res[i]=t[j]-h[i];
                i++;
            }
            else j++;
        }
        for(int i=h.size()-1,j=t.size()-1;i>=0&&j>=0;){
            if(h[i]>=t[j]){res[i]=min(res[i],h[i]-t[j]);i--;}
            else j--;
        }
    return *max_element(res.begin(),res.end());}
};
```



556. Next Greater Element III

Medium

Given a positive integer `n`, find *the smallest integer which has exactly the same digits existing in the integer* `n` *and is greater in value than* `n`. If no such positive integer exists, return `-1`.

Note that the returned integer should fit in **32-bit integer**, if there is a valid answer but it does not fit in **32-bit integer**, return `-1`.

Example 1:

```
Input: n = 12
Output: 21
```

Copy and Save Share

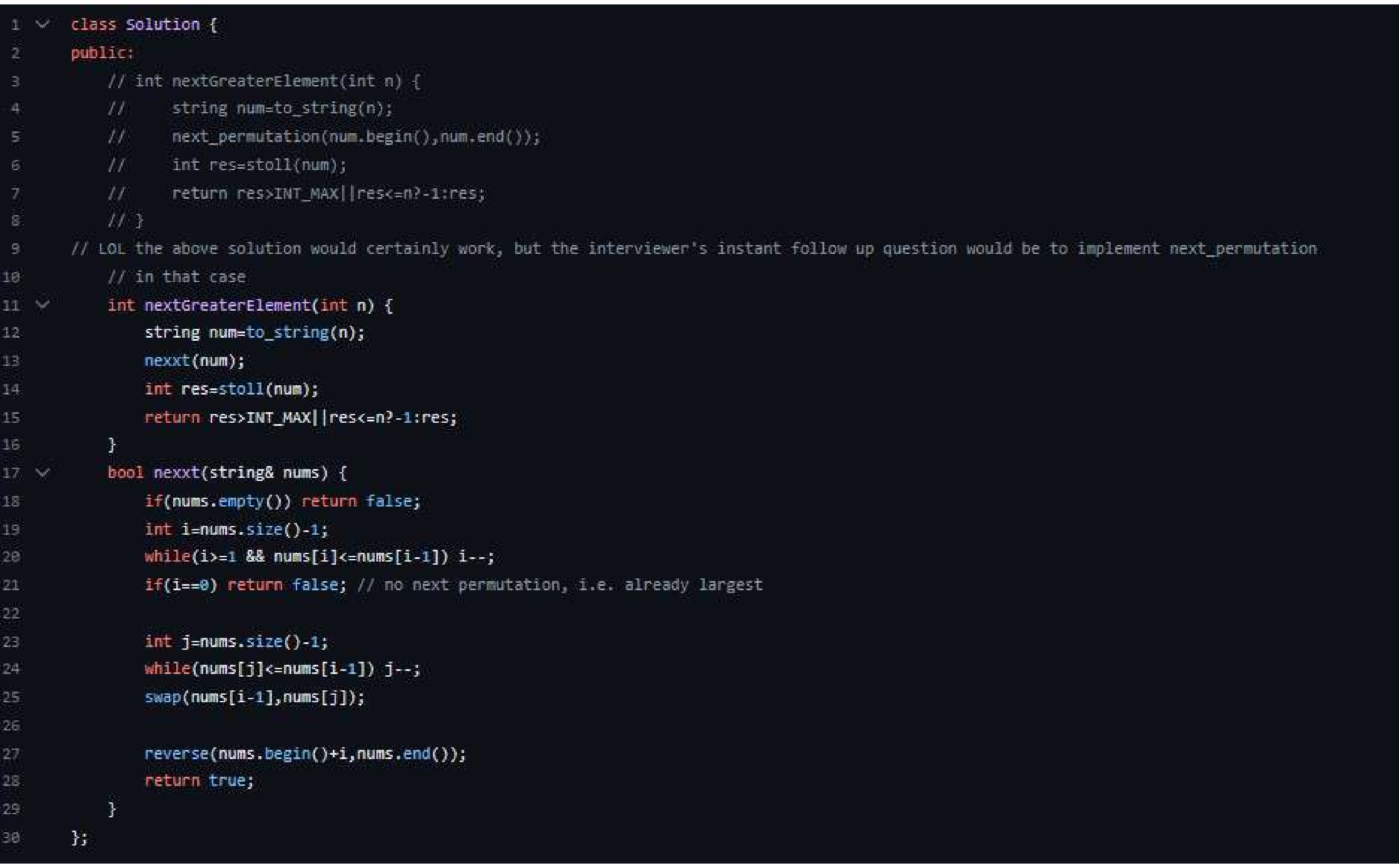
Example 2:

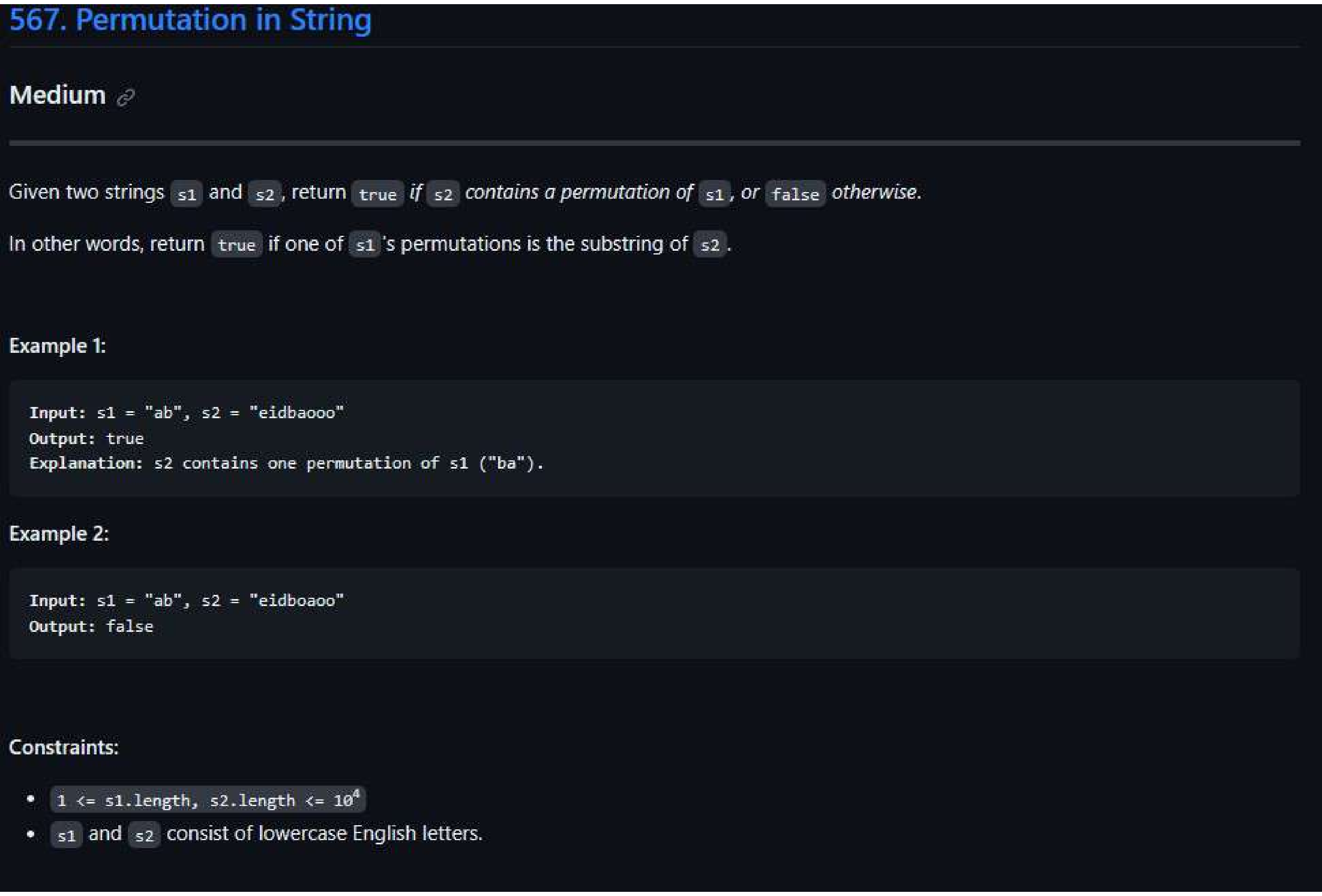
```
Input: n = 21
Output: -1
```

Copy and Save Share

Constraints:

- $1 <= n <= 2^{31} - 1$


```cpp
class Solution {
public:
    // int nextGreaterElement(int n) {
    //     string num=to_string(n);
    //     next_permutation(num.begin(),num.end());
    //     int res=stoll(num);
    //     return res>INT_MAX||res<=n?-1:res;
    // }
// LOL the above solution would certainly work, but the interviewer's instant follow up question would be to implement next_permutation
    // in that case
    int nextGreaterElement(int n) {
        string num=to_string(n);
        nexxt(num);
        int res=stoll(num);
        return res>INT_MAX||res<=n?-1:res;
    }
    bool nexxt(string& nums) {
        if(nums.empty()) return false;
        int i=nums.size()-1;
        while(i>=1 && nums[i]<=nums[i-1]) i--;
        if(i==0) return false; // no next permutation, i.e. already largest

        int j=nums.size()-1;
        while(nums[j]<=nums[i-1]) j--;
        swap(nums[i-1],nums[j]);

        reverse(nums.begin()+i,nums.end());
        return true;
    }
};
```



567. Permutation in String

Medium

Given two strings `s1` and `s2`, return `true` *if* `s2` *contains a permutation of* `s1`*, or* `false` *otherwise.*

In other words, return `true` if one of `s1`'s permutations is the substring of `s2`.

Example 1:

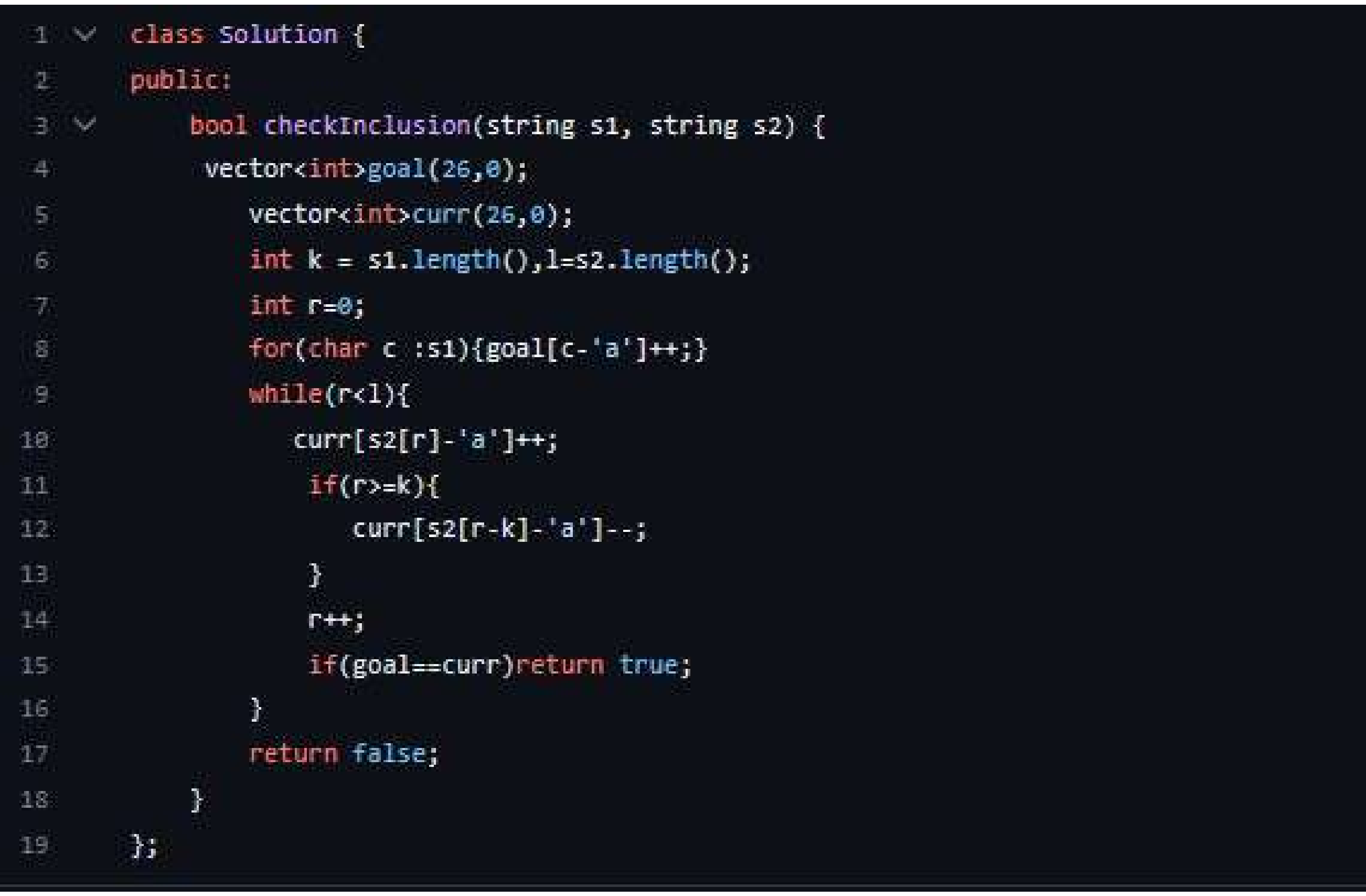
```
Input: s1 = "ab", s2 = "eidbaooo"
Output: true
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

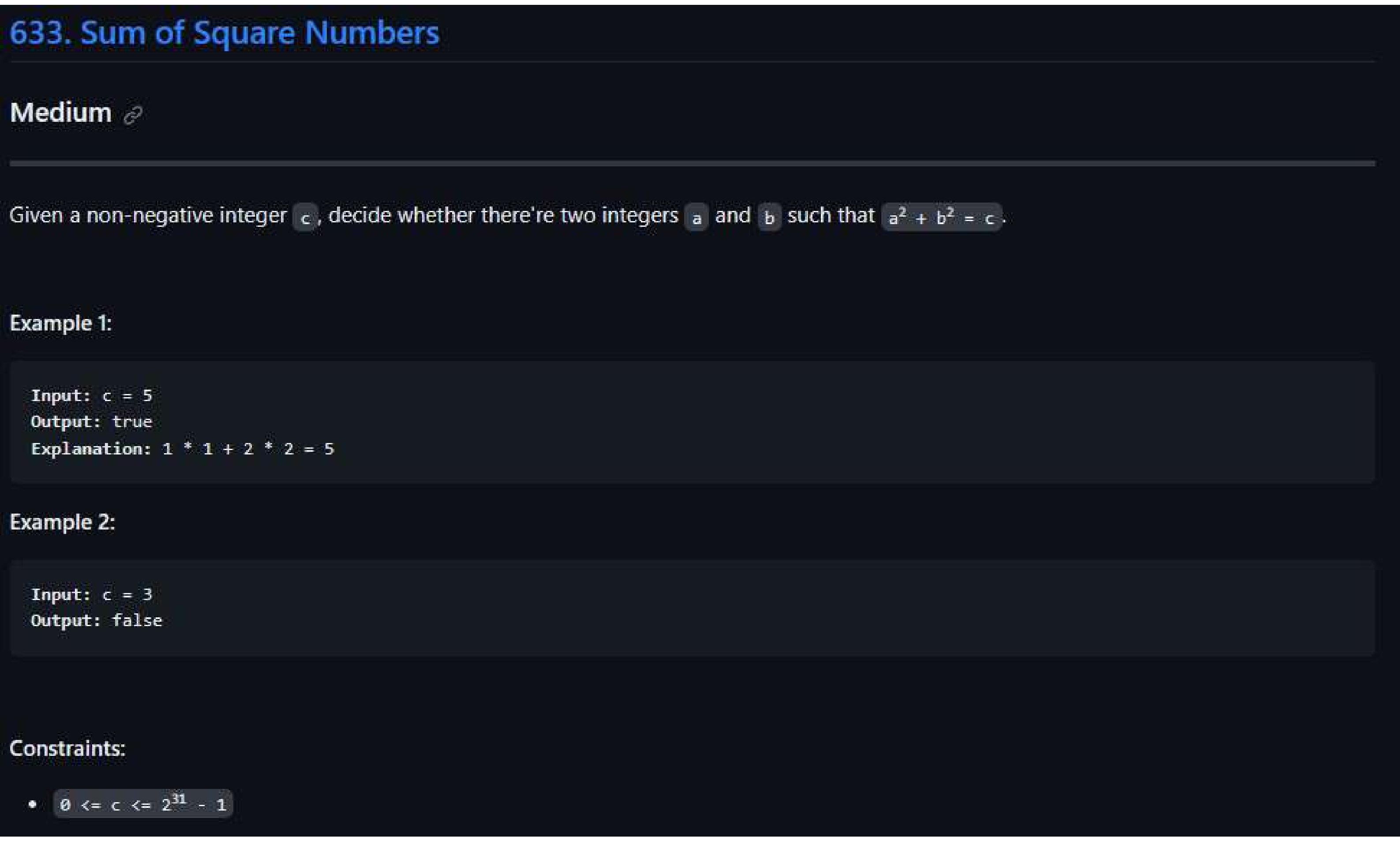
```
Input: s1 = "ab", s2 = "eidboaoo"
Output: false
```

Constraints:

- $1 <= s1.length, s2.length <= 10^4$
- `s1` and `s2` consist of lowercase English letters.



```
class Solution {
public:
    bool checkInclusion(string s1, string s2) {
     vector<int>goal(26,0);
        vector<int>curr(26,0);
        int k = s1.length(),l=s2.length();
        int r=0;
        for(char c :s1){goal[c-'a']++;}
        while(r<l){
           curr[s2[r]-'a']++;
            if(r>=k){
               curr[s2[r-k]-'a']--;
            }
            r++;
            if(goal==curr)return true;
        }
        return false;
    }
};
```



633. Sum of Square Numbers

Medium

Given a non-negative integer `c`, decide whether there're two integers `a` and `b` such that `a^2 + b^2 = c`.

Example 1:

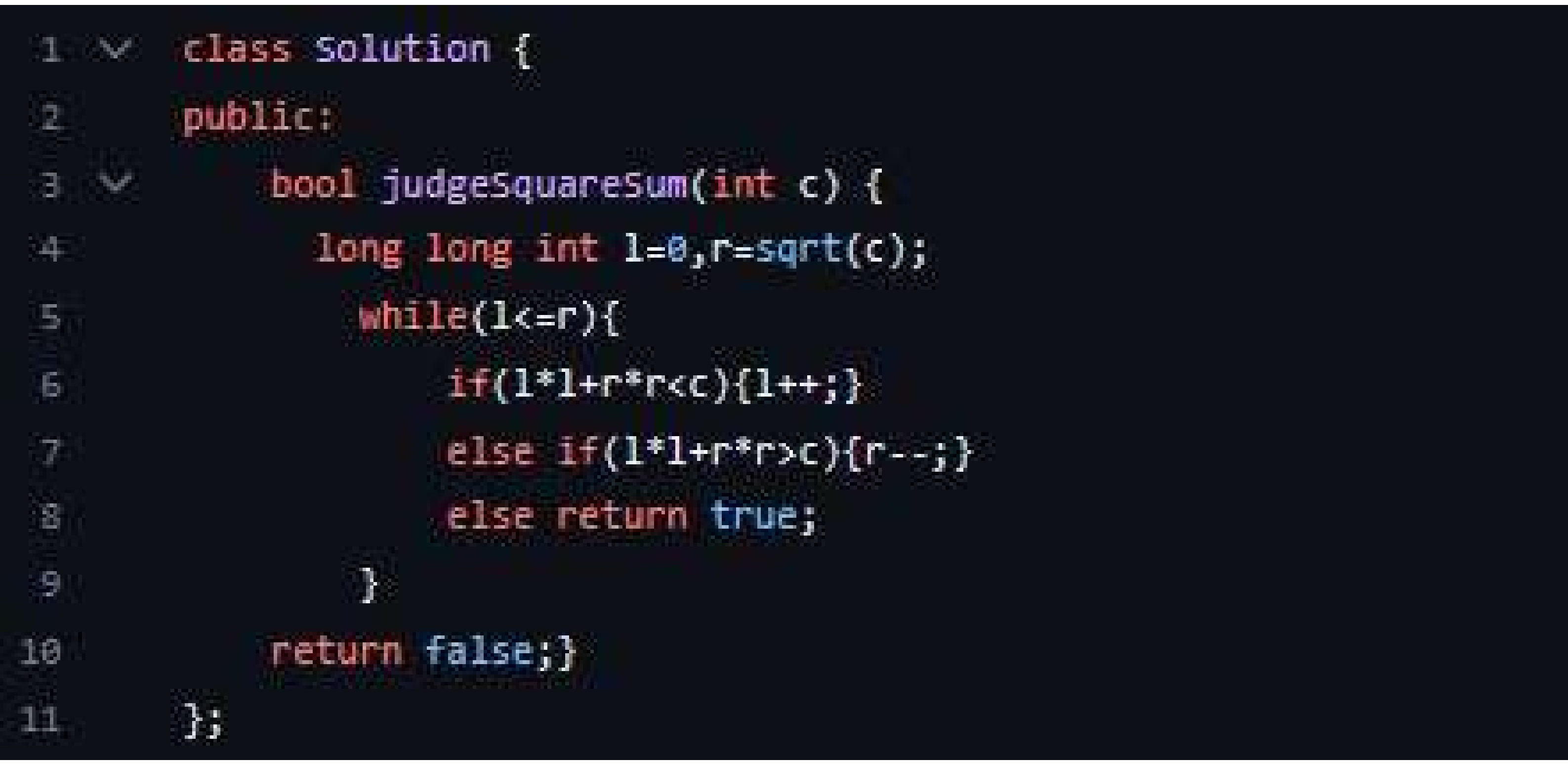
```
Input: c = 5
Output: true
Explanation: 1 * 1 + 2 * 2 = 5
```

Example 2:

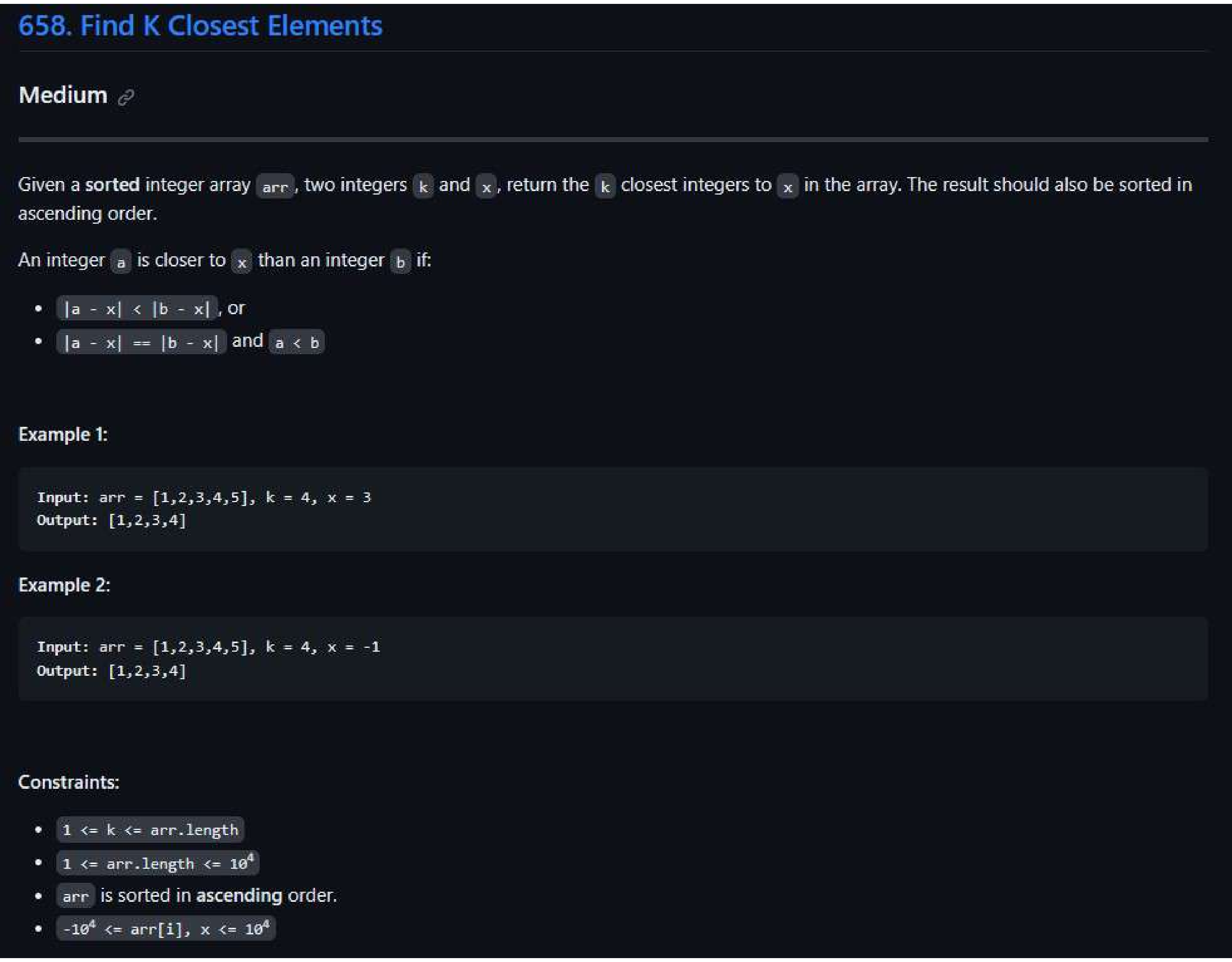
```
Input: c = 3
Output: false
```

Constraints:

- $0 <= c <= 2^{31} - 1$



```
class Solution {
public:
    bool judgeSquareSum(int c) {
      long long int l=0,r=sqrt(c);
        while(l<=r){
            if(l*l+r*r<c){l++;}
            else if(l*l+r*r>c){r--;}
            else return true;
        }
    return false;}
};
```



658. Find K Closest Elements

Medium

Given a **sorted** integer array `arr`, two integers `k` and `x`, return the `k` closest integers to `x` in the array. The result should also be sorted in ascending order.

An integer `a` is closer to `x` than an integer `b` if:

- `|a - x| < |b - x|`, or
- `|a - x| == |b - x|` and `a < b`

Example 1:

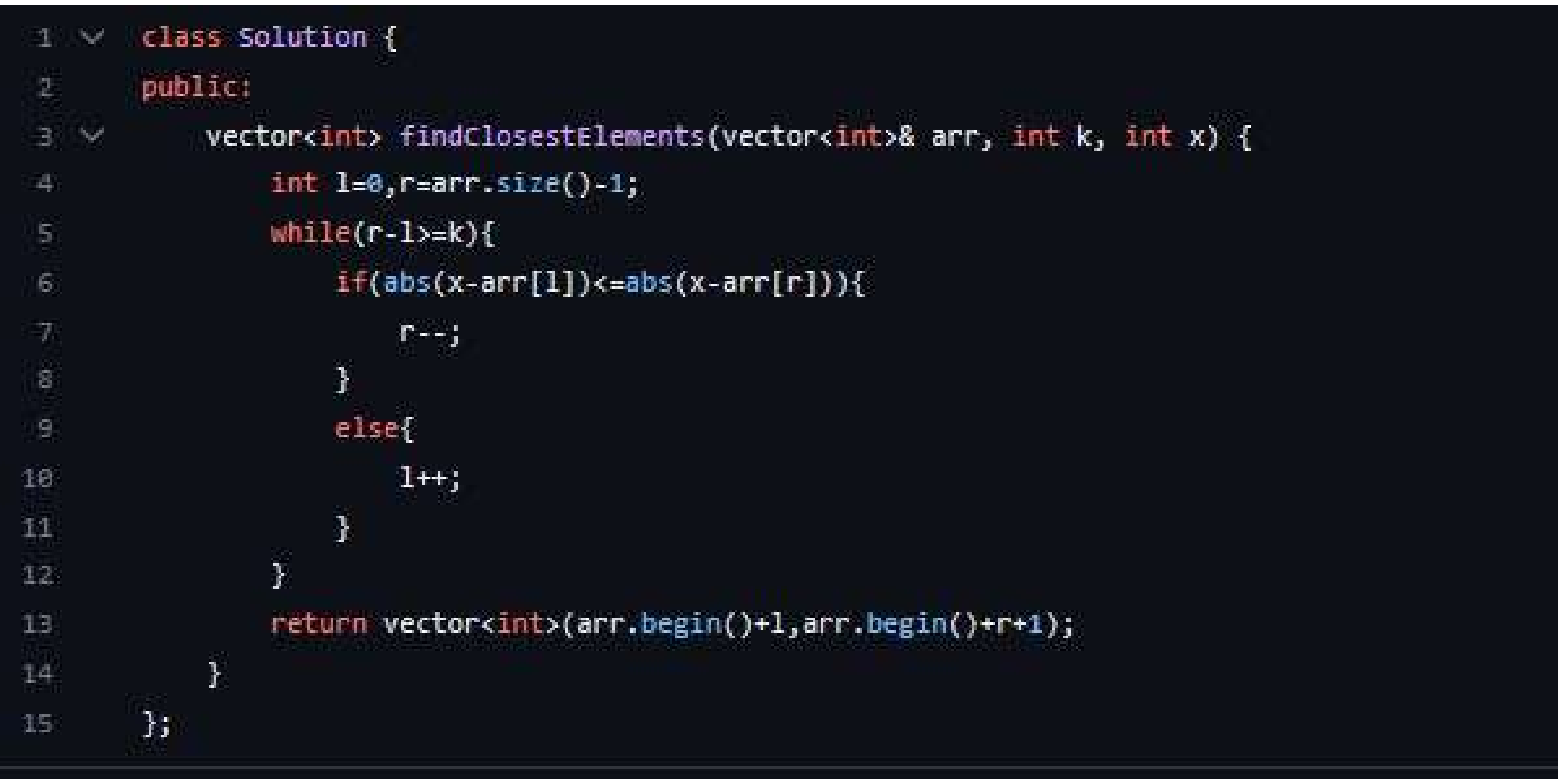
```
Input: arr = [1,2,3,4,5], k = 4, x = 3
Output: [1,2,3,4]
```

Example 2:

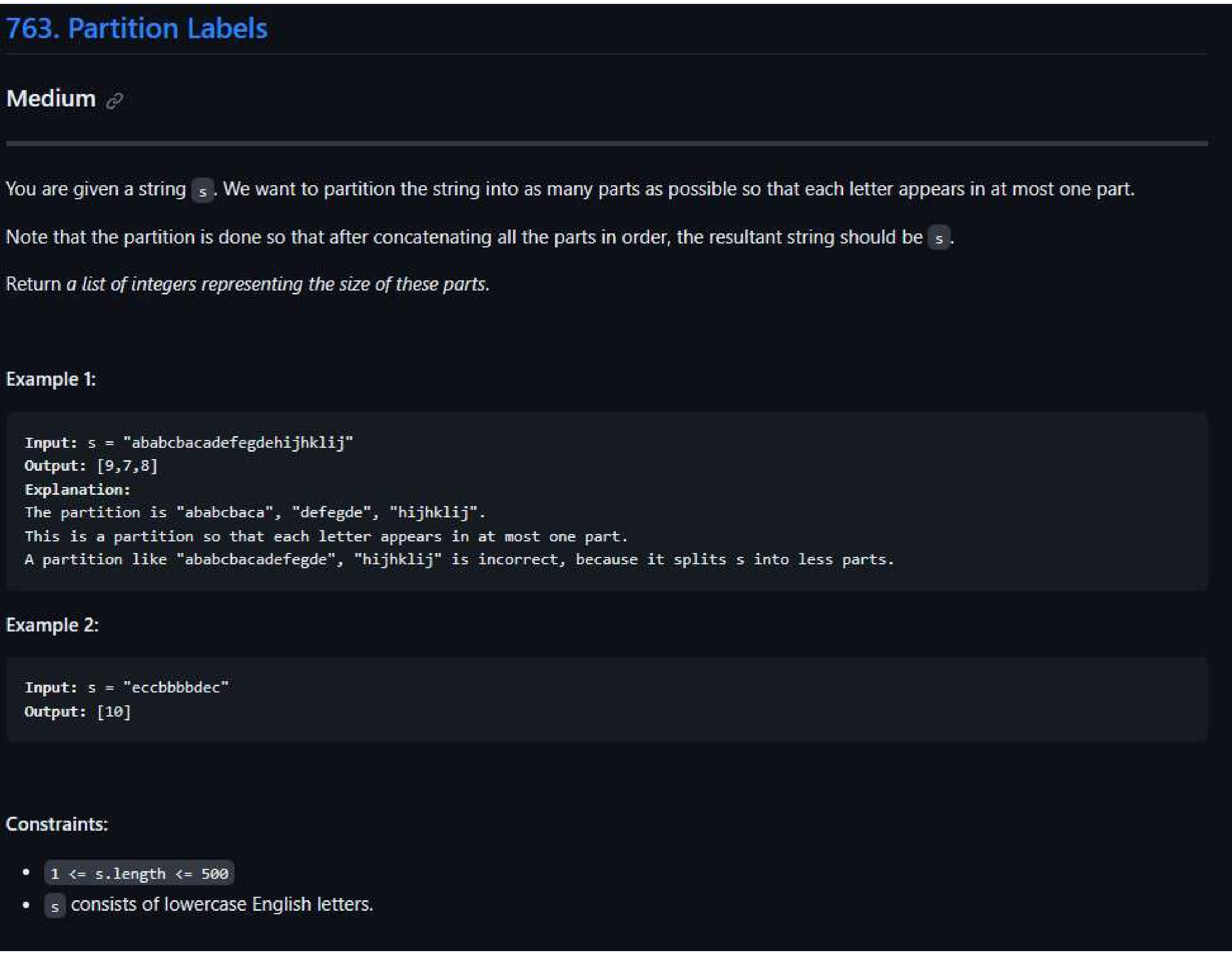
```
Input: arr = [1,2,3,4,5], k = 4, x = -1
Output: [1,2,3,4]
```

Constraints:

- `1 <= k <= arr.length`
- $1 <= arr.length <= 10^4$
- `arr` is sorted in **ascending** order.
- $-10^4 <= arr[i], x <= 10^4$



```cpp
class Solution {
public:
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
        int l=0,r=arr.size()-1;
        while(r-l>=k){
            if(abs(x-arr[l])<=abs(x-arr[r])){
                r--;
            }
            else{
                l++;
            }
        }
        return vector<int>(arr.begin()+l,arr.begin()+r+1);
    }
};
```



763. Partition Labels

Medium

You are given a string `s`. We want to partition the string into as many parts as possible so that each letter appears in at most one part.

Note that the partition is done so that after concatenating all the parts in order, the resultant string should be `s`.

Return *a list of integers representing the size of these parts.*

Example 1:

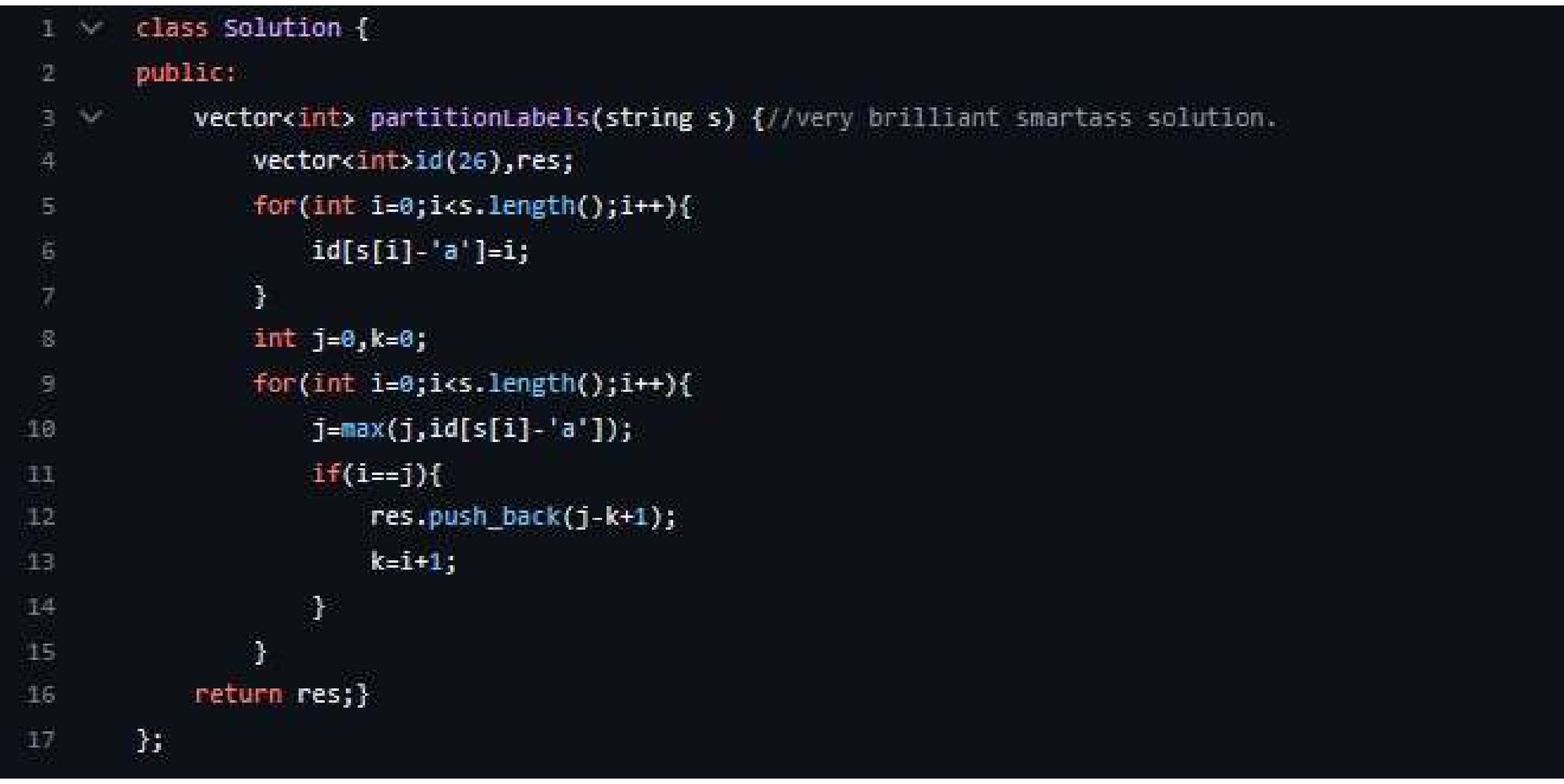
```
Input: s = "ababcbacadefegdehijhklij"
Output: [9,7,8]
Explanation:
The partition is "ababcbaca", "defegde", "hijhklij".
This is a partition so that each letter appears in at most one part.
A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it splits s into less parts.
```

Example 2:

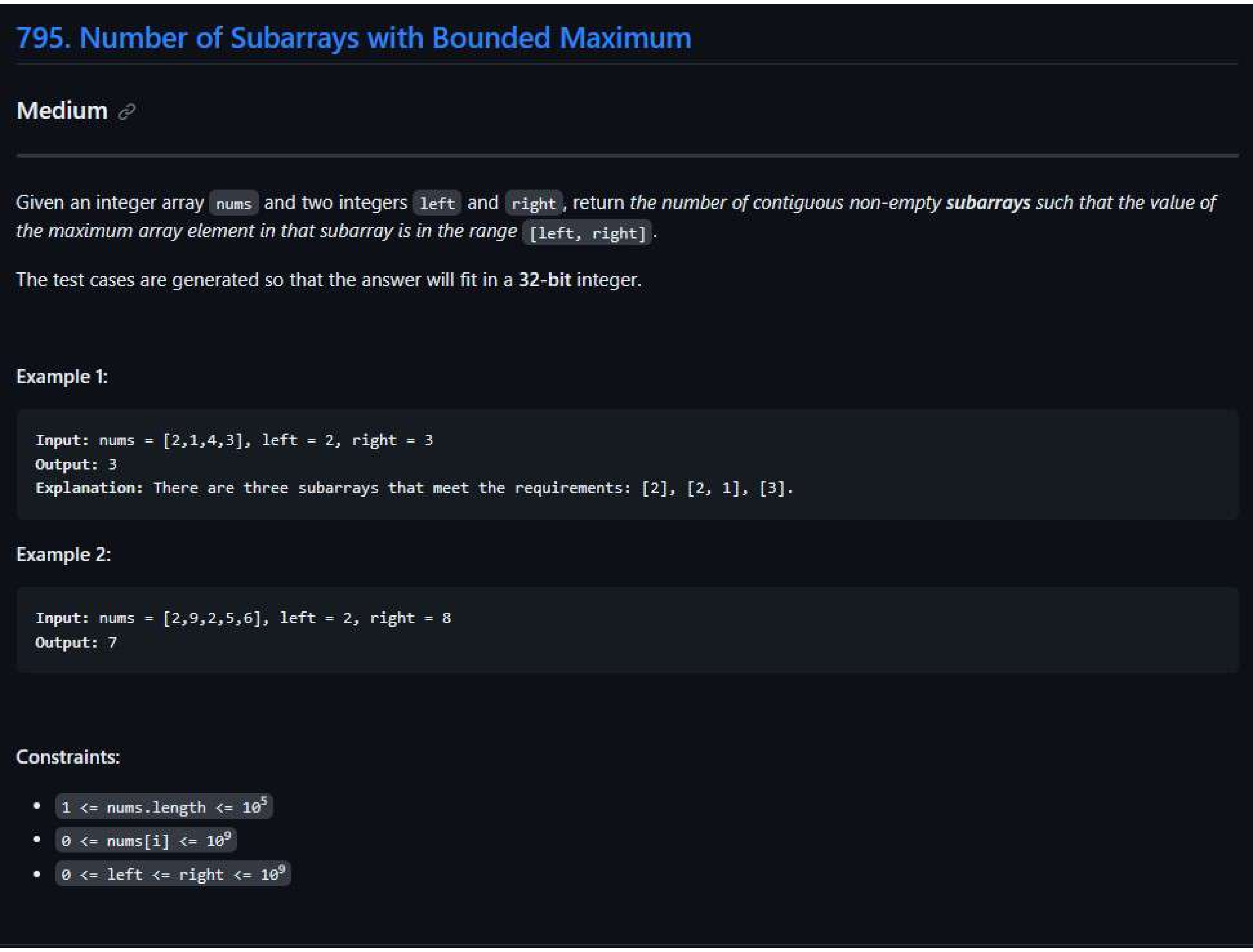
```
Input: s = "eccbbbbdec"
Output: [10]
```

Constraints:

- `1 <= s.length <= 500`
- `s` consists of lowercase English letters.



```cpp
class Solution {
public:
    vector<int> partitionLabels(string s) {//very brilliant smartass solution.
        vector<int>id(26),res;
        for(int i=0;i<s.length();i++){
            id[s[i]-'a']=i;
        }
        int j=0,k=0;
        for(int i=0;i<s.length();i++){
            j=max(j,id[s[i]-'a']);
            if(i==j){
                res.push_back(j-k+1);
                k=i+1;
            }
        }
    return res;}
};
```



795. Number of Subarrays with Bounded Maximum

Medium

Given an integer array `nums` and two integers `left` and `right`, return *the number of contiguous non-empty **subarrays** such that the value of the maximum array element in that subarray is in the range* `[left, right]`.

The test cases are generated so that the answer will fit in a **32-bit** integer.

Example 1:

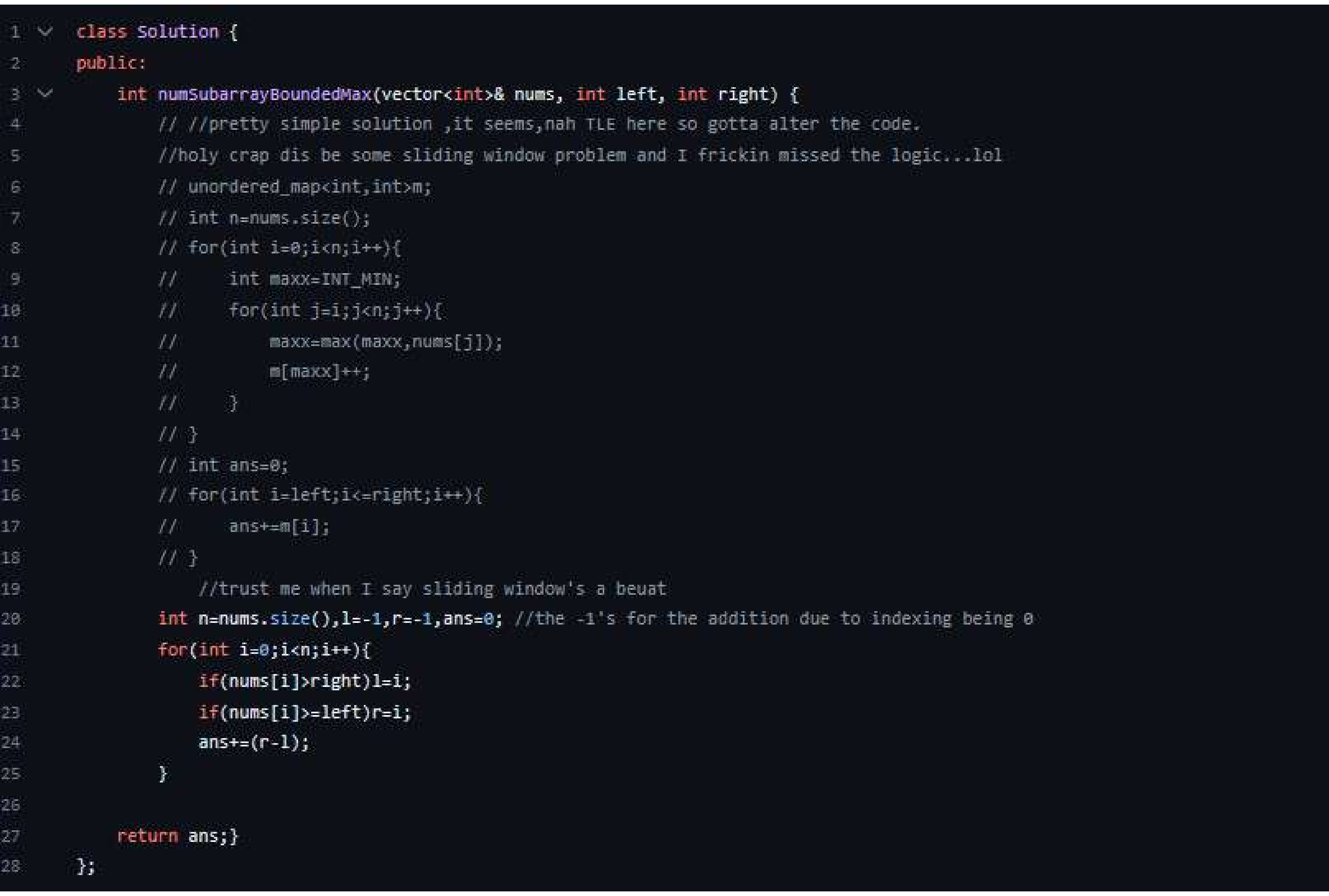
```
Input: nums = [2,1,4,3], left = 2, right = 3
Output: 3
Explanation: There are three subarrays that meet the requirements: [2], [2, 1], [3].
```

Example 2:

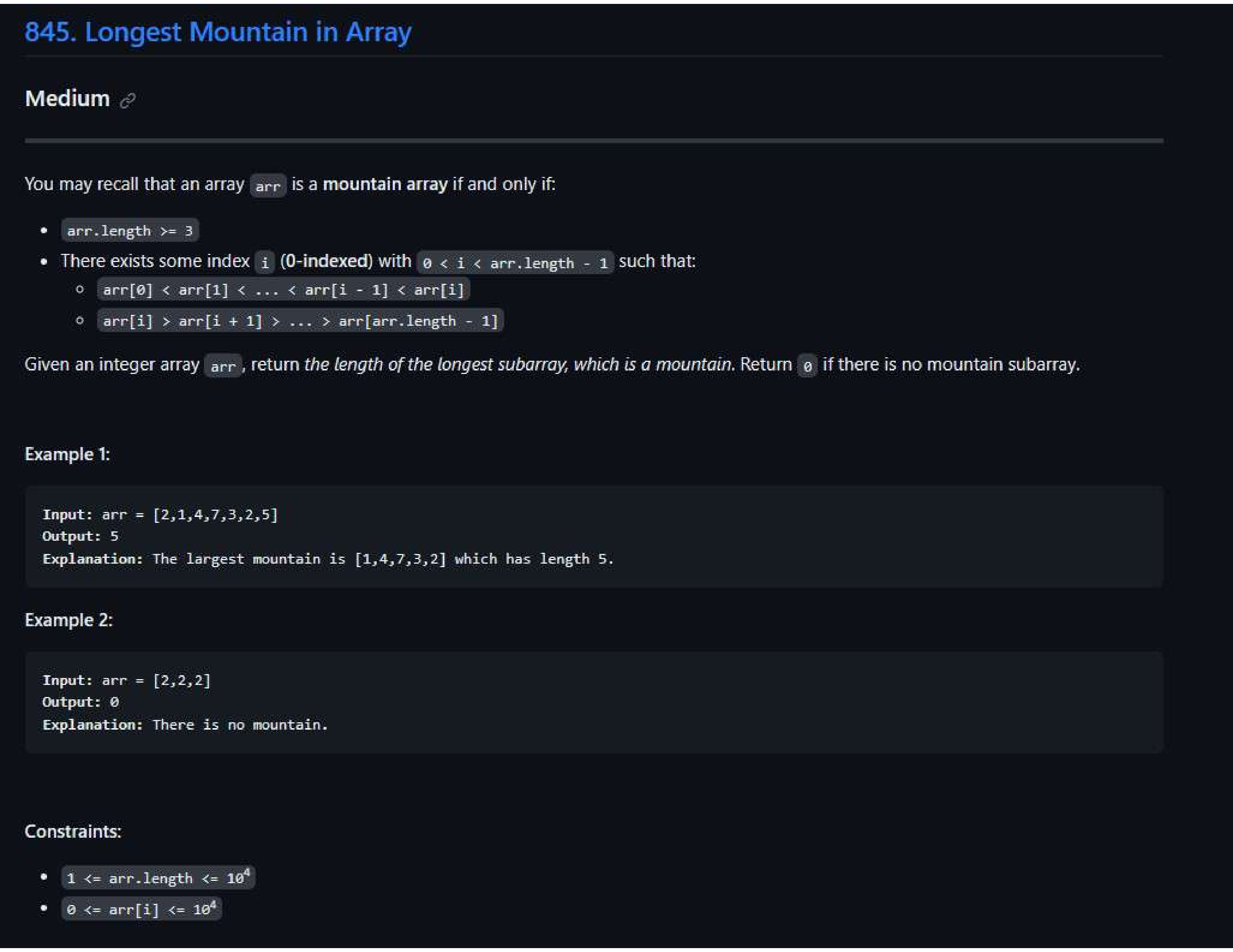
```
Input: nums = [2,9,2,5,6], left = 2, right = 8
Output: 7
```

Constraints:

- `1 <= nums.length <=` 10^5
- `0 <= nums[i] <=` 10^9
- `0 <= left <= right <=` 10^9



```cpp
class Solution {
public:
    int numSubarrayBoundedMax(vector<int>& nums, int left, int right) {
        // //pretty simple solution ,it seems,nah TLE here so gotta alter the code.
        //holy crap dis be some sliding window problem and I frickin missed the logic...lol
        // unordered_map<int,int>m;
        // int n=nums.size();
        // for(int i=0;i<n;i++){
        //     int maxx=INT_MIN;
        //     for(int j=i;j<n;j++){
        //         maxx=max(maxx,nums[j]);
        //         m[maxx]++;
        //     }
        // }
        // int ans=0;
        // for(int i=left;i<=right;i++){
        //     ans+=m[i];
        // }
            //trust me when I say sliding window's a beuat
        int n=nums.size(),l=-1,r=-1,ans=0; //the -1's for the addition due to indexing being 0
        for(int i=0;i<n;i++){
            if(nums[i]>right)l=i;
            if(nums[i]>=left)r=i;
            ans+=(r-l);
        }

    return ans;}
};
```



845. Longest Mountain in Array

Medium

You may recall that an array `arr` is a **mountain array** if and only if:

- `arr.length >= 3`
- There exists some index `i` (**0-indexed**) with `0 < i < arr.length - 1` such that:
    - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
    - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

Given an integer array `arr`, return *the length of the longest subarray, which is a mountain*. Return `0` if there is no mountain subarray.

Example 1:

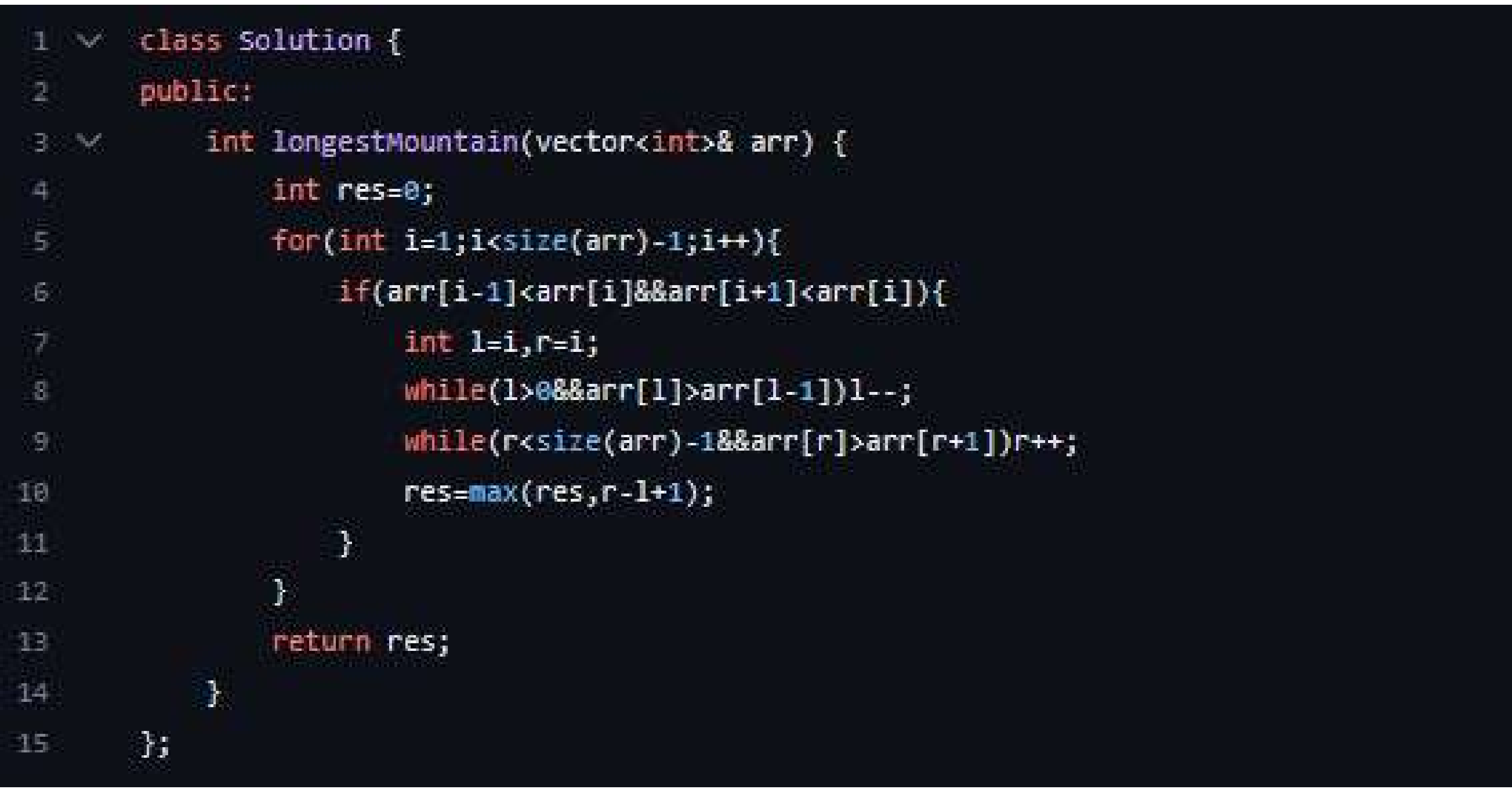
```
Input: arr = [2,1,4,7,3,2,5]
Output: 5
Explanation: The largest mountain is [1,4,7,3,2] which has length 5.
```

Example 2:

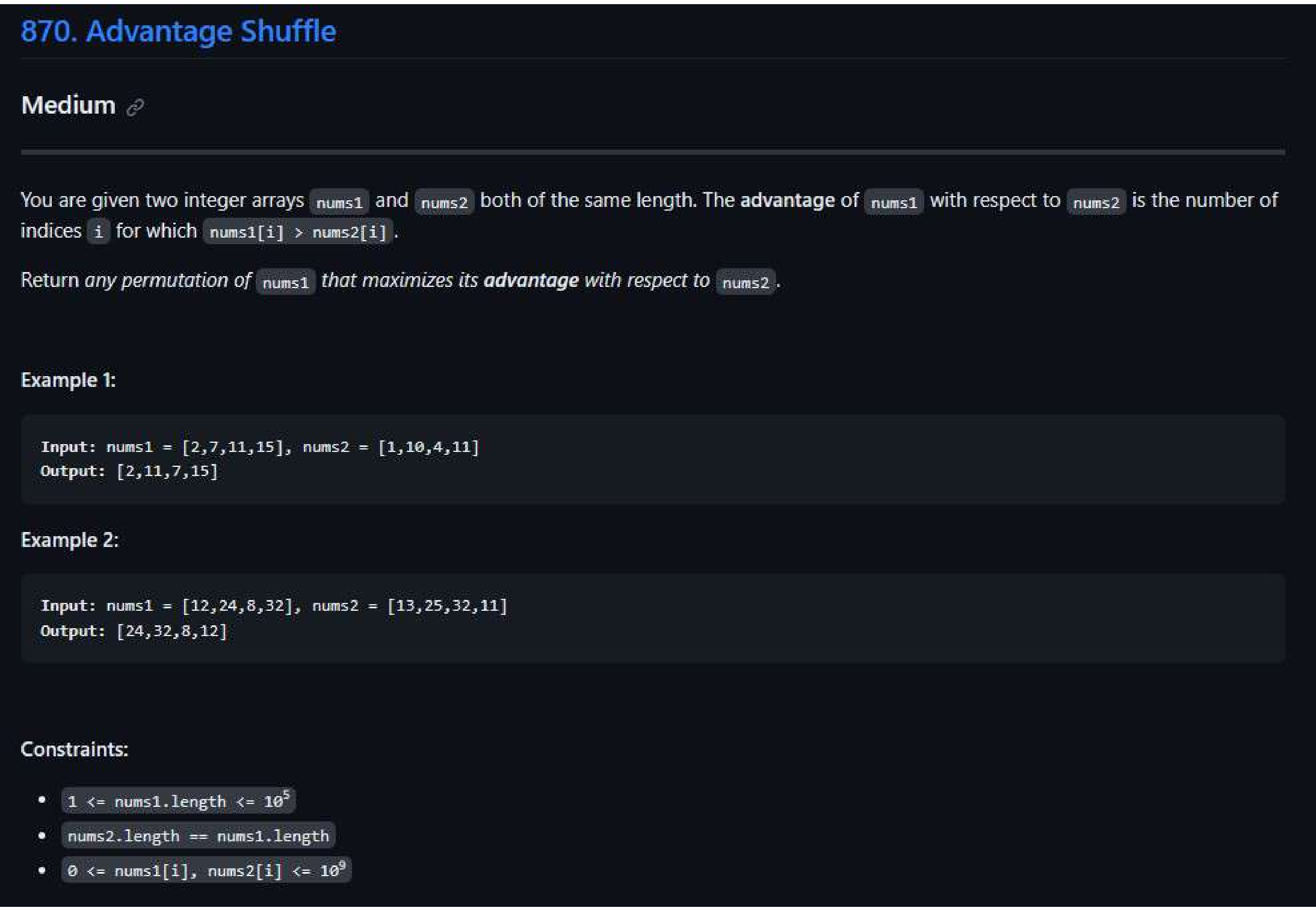
```
Input: arr = [2,2,2]
Output: 0
Explanation: There is no mountain.
```

Constraints:

- `1 <= arr.length <=` 10^4
- `0 <= arr[i] <=` 10^4



```cpp
class Solution {
public:
    int longestMountain(vector<int>& arr) {
        int res=0;
        for(int i=1;i<size(arr)-1;i++){
            if(arr[i-1]<arr[i]&&arr[i+1]<arr[i]){
                int l=i,r=i;
                while(l>0&&arr[l]>arr[l-1])l--;
                while(r<size(arr)-1&&arr[r]>arr[r+1])r++;
                res=max(res,r-l+1);
            }
        }
        return res;
    }
};
```



870. Advantage Shuffle

Medium

You are given two integer arrays `nums1` and `nums2` both of the same length. The **advantage** of `nums1` with respect to `nums2` is the number of indices `i` for which `nums1[i] > nums2[i]`.

Return *any permutation of* `nums1` *that maximizes its* ***advantage*** *with respect to* `nums2`.

Example 1:

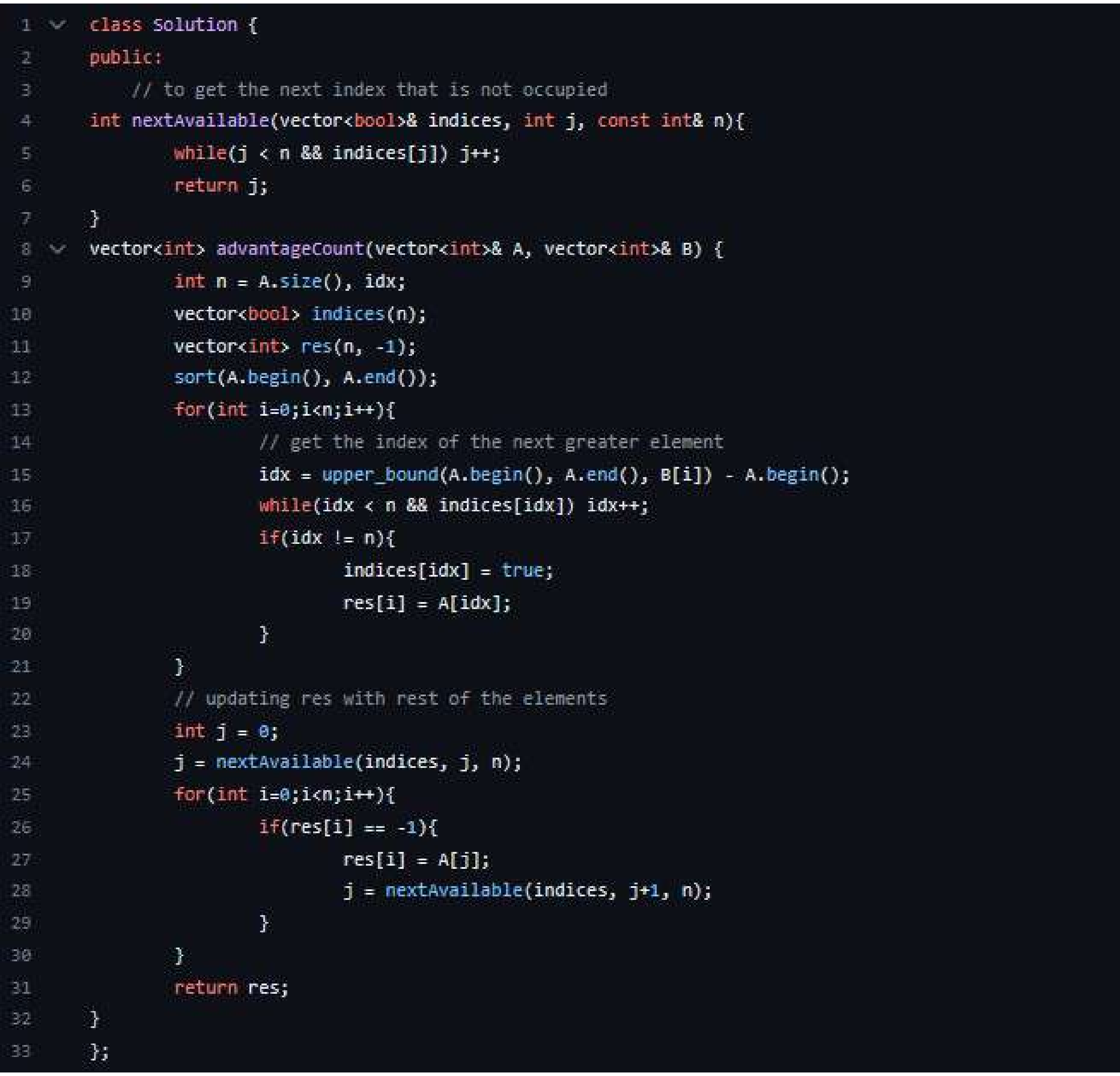
```
Input: nums1 = [2,7,11,15], nums2 = [1,10,4,11]
Output: [2,11,7,15]
```

Example 2:

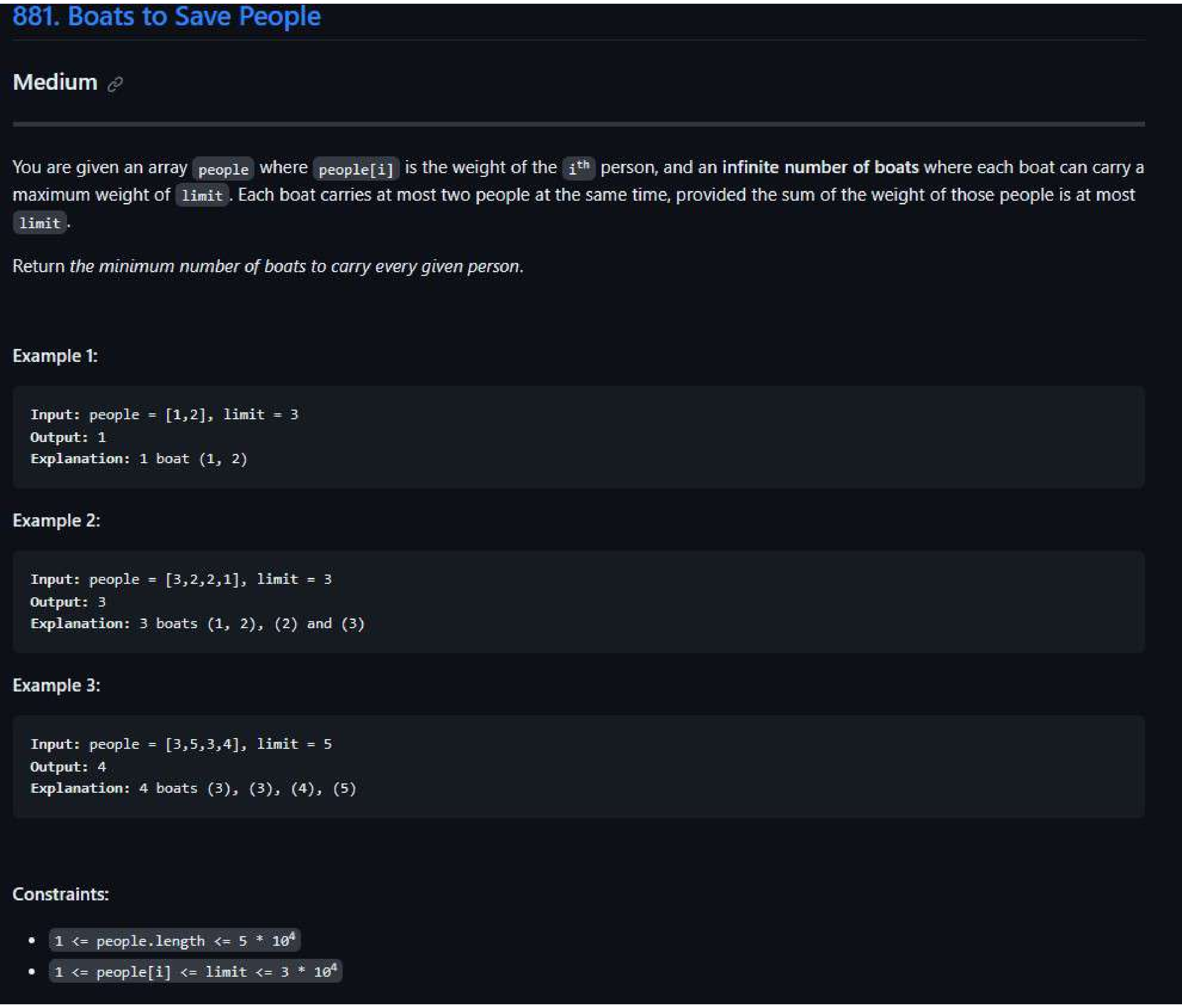
```
Input: nums1 = [12,24,8,32], nums2 = [13,25,32,11]
Output: [24,32,8,12]
```

Constraints:

- $1 <= \text{nums1.length} <= 10^5$
- `nums2.length == nums1.length`
- $0 <= \text{nums1[i], nums2[i]} <= 10^9$



```cpp
class Solution {
public:
    // to get the next index that is not occupied
int nextAvailable(vector<bool>& indices, int j, const int& n){
        while(j < n && indices[j]) j++;
        return j;
}
vector<int> advantageCount(vector<int>& A, vector<int>& B) {
        int n = A.size(), idx;
        vector<bool> indices(n);
        vector<int> res(n, -1);
        sort(A.begin(), A.end());
        for(int i=0;i<n;i++){
                // get the index of the next greater element
                idx = upper_bound(A.begin(), A.end(), B[i]) - A.begin();
                while(idx < n && indices[idx]) idx++;
                if(idx != n){
                        indices[idx] = true;
                        res[i] = A[idx];
                }
        }
        // updating res with rest of the elements
        int j = 0;
        j = nextAvailable(indices, j, n);
        for(int i=0;i<n;i++){
                if(res[i] == -1){
                        res[i] = A[j];
                        j = nextAvailable(indices, j+1, n);
                }
        }
        return res;
}
};
```



881. Boats to Save People

Medium

You are given an array `people` where `people[i]` is the weight of the i^{th} person, and an **infinite number of boats** where each boat can carry a maximum weight of `limit`. Each boat carries at most two people at the same time, provided the sum of the weight of those people is at most `limit`.

Return *the minimum number of boats to carry every given person.*

Example 1:

```
Input: people = [1,2], limit = 3
Output: 1
Explanation: 1 boat (1, 2)
```

Example 2:

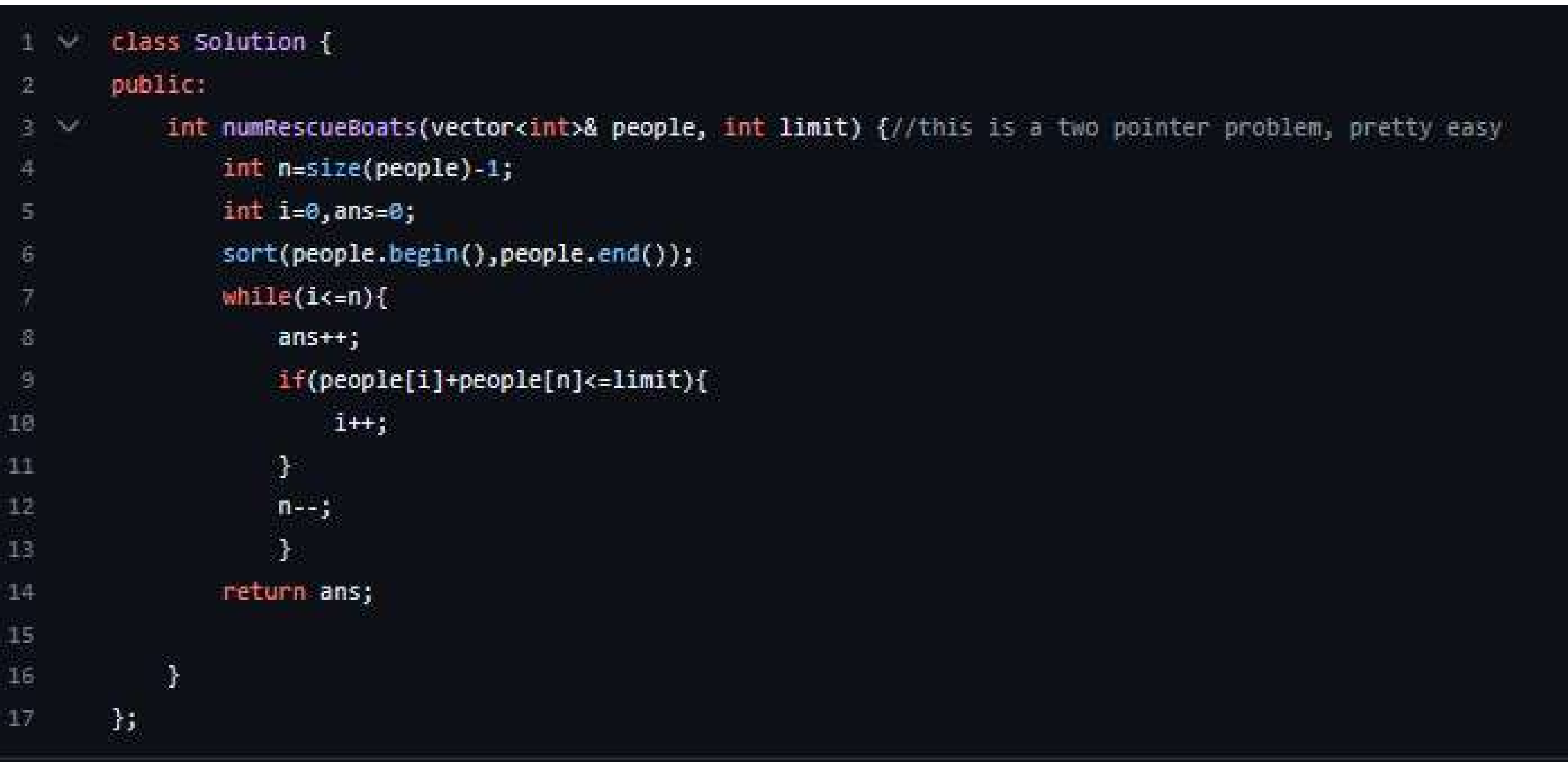
```
Input: people = [3,2,2,1], limit = 3
Output: 3
Explanation: 3 boats (1, 2), (2) and (3)
```

Example 3:

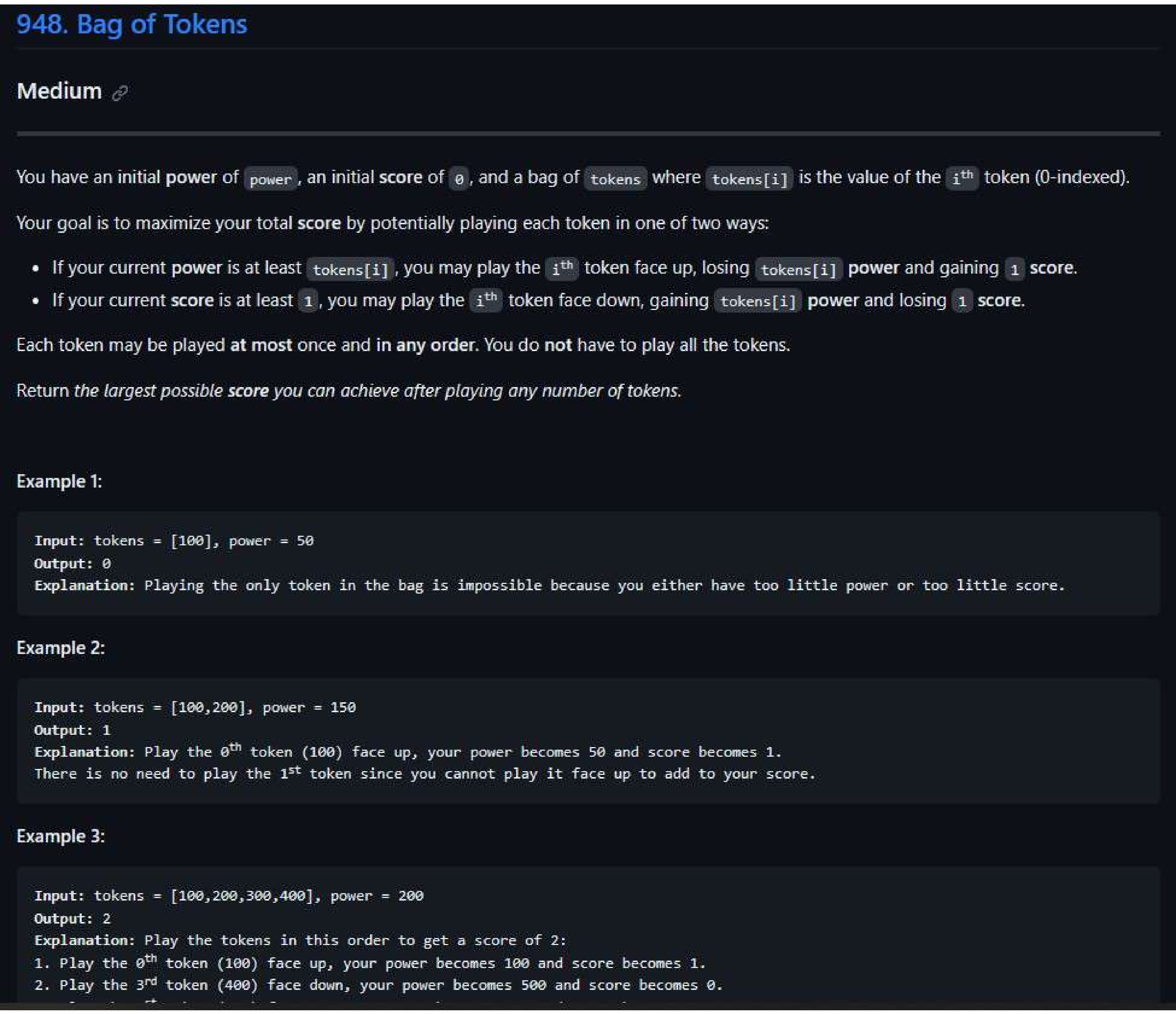
```
Input: people = [3,5,3,4], limit = 5
Output: 4
Explanation: 4 boats (3), (3), (4), (5)
```

Constraints:

- `1 <= people.length <= 5 * ` 10^4
- `1 <= people[i] <= limit <= 3 * ` 10^4



```cpp
class Solution {
public:
    int numRescueBoats(vector<int>& people, int limit) {//this is a two pointer problem, pretty easy
        int n=size(people)-1;
        int i=0,ans=0;
        sort(people.begin(),people.end());
        while(i<=n){
            ans++;
            if(people[i]+people[n]<=limit){
                i++;
            }
            n--;
            }
        return ans;

    }
};
```



948. Bag of Tokens

Medium

You have an initial **power** of `power`, an initial **score** of `0`, and a bag of `tokens` where `tokens[i]` is the value of the `ith` token (0-indexed).

Your goal is to maximize your total **score** by potentially playing each token in one of two ways:

- If your current **power** is at least `tokens[i]`, you may play the `ith` token face up, losing `tokens[i]` **power** and gaining `1` **score**.
- If your current **score** is at least `1`, you may play the `ith` token face down, gaining `tokens[i]` **power** and losing `1` **score**.

Each token may be played **at most** once and **in any order**. You do **not** have to play all the tokens.

Return *the largest possible **score** you can achieve after playing any number of tokens.*

Example 1:

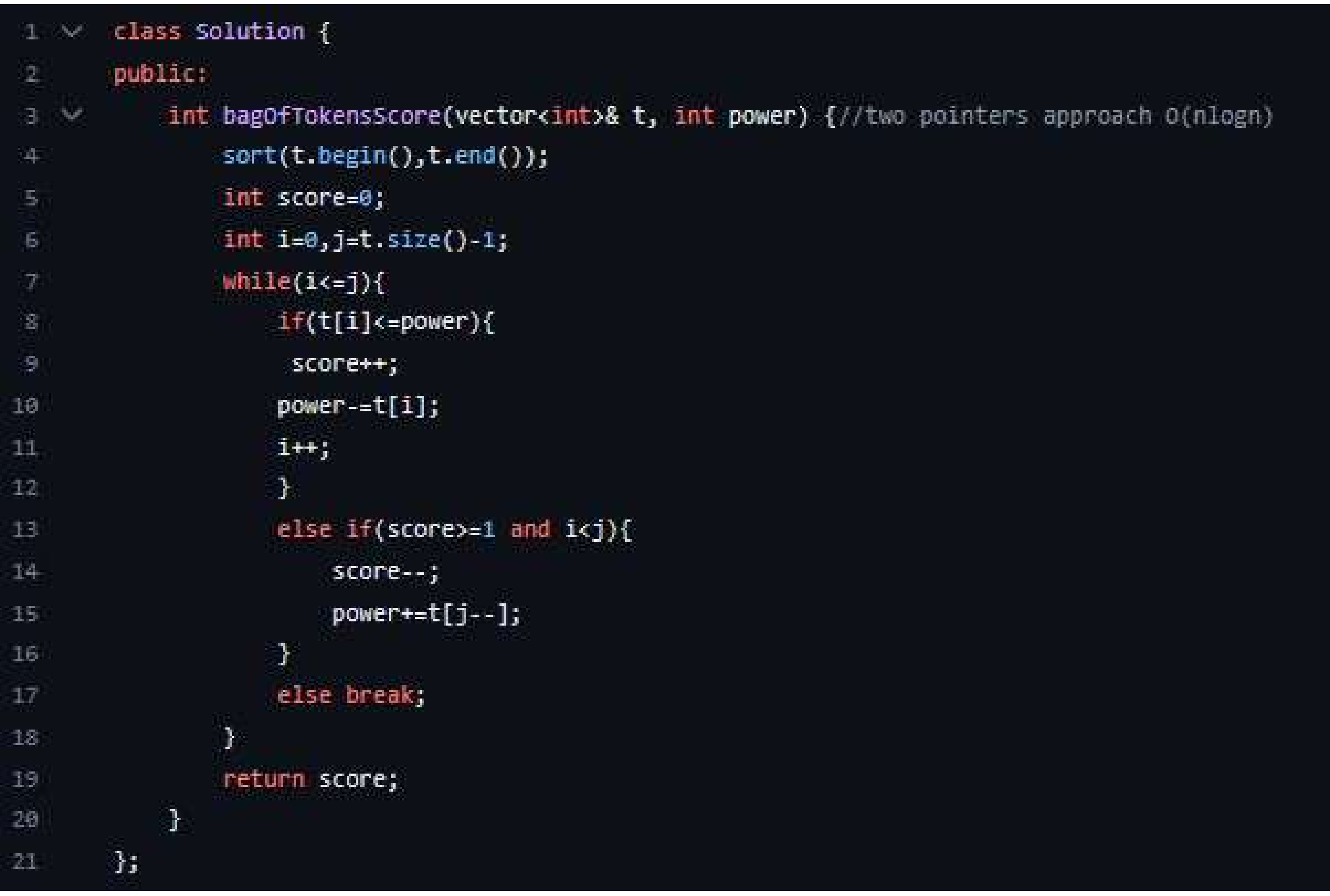
```
Input: tokens = [100], power = 50
Output: 0
Explanation: Playing the only token in the bag is impossible because you either have too little power or too little score.
```

Example 2:

```
Input: tokens = [100,200], power = 150
Output: 1
Explanation: Play the 0th token (100) face up, your power becomes 50 and score becomes 1.
There is no need to play the 1st token since you cannot play it face up to add to your score.
```

Example 3:

```
Input: tokens = [100,200,300,400], power = 200
Output: 2
Explanation: Play the tokens in this order to get a score of 2:
1. Play the 0th token (100) face up, your power becomes 100 and score becomes 1.
2. Play the 3rd token (400) face down, your power becomes 500 and score becomes 0.
```



```cpp
class Solution {
public:
    int bagOfTokensScore(vector<int>& t, int power) {//two pointers approach O(nlogn)
        sort(t.begin(),t.end());
        int score=0;
        int i=0,j=t.size()-1;
        while(i<=j){
            if(t[i]<=power){
             score++;
            power-=t[i];
            i++;
            }
            else if(score>=1 and i<j){
                score--;
                power+=t[j--];
            }
            else break;
        }
        return score;
    }
};
```