



## **Contents**

- 1. Subarray Product Less Than K - Medium**
- 2. Sort Colors - Medium**
- 3. Container With Most Water - Medium**
- 4. Longest Word in Dictionary - Medium**
- 5. Maximum XOR of Two Numbers in an Array - Medium**
- 6. Split a String Into the Max Number of Unique Substrings - Medium**
- 7. Maximum Number of Vowels in a Substring of Given Length - Medium**
- 8. Max Consecutive Ones III - Medium**
- 9. Longest Subarray of 1's After Deleting One Element - Medium**
- 10. First Missing Positive - Hard**
- 11. Sudoku Solver - Hard**
- 12. N-Queens - Hard**
- 13. Reverse Nodes in k-Group - Hard**
- 14. Merge k Sorted Lists - Hard**

## Subarray Product Less Than K

Given an array of integers `nums` and an integer `k`, return *the number of contiguous subarrays where the product of all the elements in the subarray is strictly less than k*.

### Example 1:

**Input:** `nums = [10,5,2,6]`, `k = 100` **Output:** 8 **Explanation:** The 8 subarrays that have product less than 100 are: `[10]`, `[5]`, `[2]`, `[6]`, `[10, 5]`, `[5, 2]`, `[2, 6]`, `[5, 2, 6]` Note that `[10, 5, 2]` is not included as the product of 100 is not strictly less than `k`.

### Example 2:

**Input:** `nums = [1,2,3]`, `k = 0` **Output:** 0

### Constraints:

- $1 \leq \text{nums.length} \leq 3 \times 10^4$
  - $1 \leq \text{nums}[i] \leq 1000$
  - $0 \leq k \leq 10^6$
- ```
class Solution { public: int numSubarrayProductLessThanK(vector& nums, int k) { } };
```

## Sort Colors

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.

**Follow up:** Could you come up with a one-pass algorithm using only constant extra space?

```
class Solution { public: void sortColors(vector& nums) { } };
```

## Container With Most Water

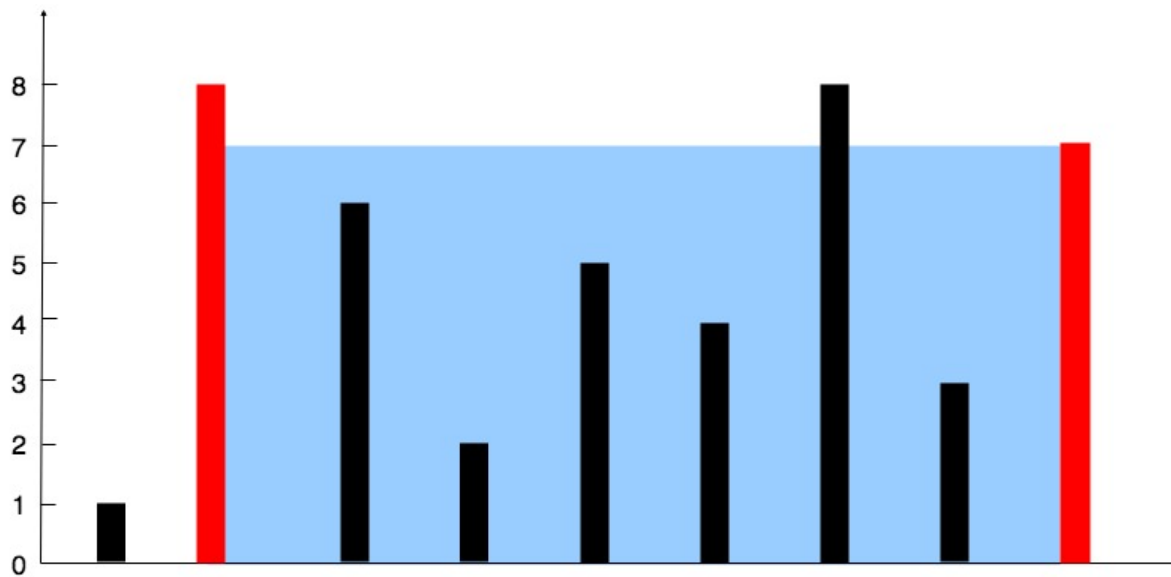
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the  $i^{\text{th}}$  line are  $(i, 0)$  and  $(i, \text{height}[i])$ .

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) the container can contain is 49.

### Example 2:

**Input:** `height = [1,1]` **Output:** 1

### Constraints:

- `n == height.length`
- `2 <= n <= 10^5`
- `0 <= height[i] <= 10^4`

```
class Solution { public: int maxArea(vector& height) { } };
```

## Longest Word in Dictionary

Given an array of strings `words` representing an English Dictionary, return *the longest word in words that can be built one character at a time by other words in words*.

If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

Note that the word should be built from left to right with each additional character being added to the end of a previous word.

### Example 1:

**Input:** `words = ["w","wo","wor","worl","world"]` **Output:** `"world"` **Explanation:** The word "world" can be built one character at a time by "w", "wo", "wor", and "worl".

### Example 2:

**Input:** `words = ["a","banana","app","appl","ap","apply","apple"]` **Output:** `"apple"` **Explanation:** Both "apply" and "apple" can be built from other words in the dictionary. However, "apple" is lexicographically smaller than "apply".

### Constraints:

- `1 <= words.length <= 1000`
- `1 <= words[i].length <= 30`
- `words[i]` consists of lowercase English letters.

```
class Solution { public: string longestWord(vector& words) {} };
```

## Maximum XOR of Two Numbers in an Array

Given an integer array `nums`, return *the maximum result of* `nums[i] XOR nums[j]`, where  $0 \leq i \leq j < n$ .

### Example 1:

**Input:** `nums = [3,10,5,25,2,8]` **Output:** 28 **Explanation:** The maximum result is 5 XOR 25 = 28.

### Example 2:

**Input:** `nums = [14,70,53,83,49,91,36,80,92,51,66,70]` **Output:** 127

### Constraints:

- $1 \leq \text{nums.length} \leq 2 \cdot 10^5$
- $0 \leq \text{nums}[i] \leq 2^{31} - 1$

```
class Solution { public: int findMaximumXOR(vector& nums) { } };
```

## Split a String Into the Max Number of Unique Substrings

Given a string `s`, return *the maximum number of unique substrings that the given string can be split into*.

You can split string `s` into any list of **non-empty substrings**, where the concatenation of the substrings forms the original string. However, you must split the substrings such that all of them are **unique**.

A **substring** is a contiguous sequence of characters within a string.

### Example 1:

**Input:** `s = "ababccc"` **Output:** 5 **Explanation:** One way to split maximally is `['a', 'b', 'ab', 'c', 'cc']`. Splitting like `['a', 'b', 'a', 'b', 'c', 'cc']` is not valid as you have 'a' and 'b' multiple times.

### Example 2:

**Input:** `s = "aba"` **Output:** 2 **Explanation:** One way to split maximally is `['a', 'ba']`.

### Example 3:

**Input:** `s = "aa"` **Output:** 1 **Explanation:** It is impossible to split the string any further.

### Constraints:

`1 <= s.length <= 16`

`s` contains only lower case English letters.

```
class Solution { public: int maxUniqueSplit(string s) { } };
```



## Maximum Number of Vowels in a Substring of Given Length

Given a string `s` and an integer `k`, return *the maximum number of vowel letters in any substring of `s` with length `k`*.

**Vowel letters** in English are 'a', 'e', 'i', 'o', and 'u'.

### Example 1:

**Input:** `s = "abciidef", k = 3` **Output:** 3 **Explanation:** The substring "iii" contains 3 vowel letters.

### Example 2:

**Input:** `s = "aeiou", k = 2` **Output:** 2 **Explanation:** Any substring of length 2 contains 2 vowels.

### Example 3:

**Input:** `s = "leetcode", k = 3` **Output:** 2 **Explanation:** "lee", "eet" and "ode" contain 2 vowels.

### Constraints:

- $1 \leq s.length \leq 10^5$
- `s` consists of lowercase English letters.
- $1 \leq k \leq s.length$

```
class Solution { public: int maxVowels(string s, int k) {} };
```

## Max Consecutive Ones III

Given a binary array `nums` and an integer `k`, return *the maximum number of consecutive 1's in the array if you can flip at most `k` 0's*.

### Example 1:

**Input:** `nums = [1,1,1,0,0,0,1,1,1,0]`, `k = 2` **Output:** 6 **Explanation:** `[1,1,1,0,0,1,1,1,1,1]` Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

### Example 2:

**Input:** `nums = [0,0,1,1,0,0,1,1,0,1,1,0,0,0,1,1,1,1]`, `k = 3` **Output:** 10 **Explanation:** `[0,0,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]` Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

### Constraints:

- `1 <= nums.length <= 105`
  - `nums[i]` is either 0 or 1.
  - `0 <= k <= nums.length`
- ```
class Solution { public: int longestOnes(vector& nums, int k) {} };
```

## Longest Subarray of 1's After Deleting One Element

Given a binary array `nums`, you should delete one element from it.

Return *the size of the longest non-empty subarray containing only 1's in the resulting array*. Return 0 if there is no such subarray.

### Example 1:

**Input:** `nums = [1,1,0,1]` **Output:** 3 **Explanation:** After deleting the number in position 2, `[1,1,1]` contains 3 numbers with value of 1's.

### Example 2:

**Input:** `nums = [0,1,1,1,0,1,1,0,1]` **Output:** 5 **Explanation:** After deleting the number in position 4, `[0,1,1,1,1,0,1]` longest subarray with value of 1's is `[1,1,1,1,1]`.

### Example 3:

**Input:** `nums = [1,1,1]` **Output:** 2 **Explanation:** You must delete one element.

### Constraints:

- `1 <= nums.length <= 105`
- `nums[i]` is either 0 or 1.

```
class Solution { public: int longestSubarray(vector& nums) {} };
```

## First Missing Positive

Given an unsorted integer array `nums`. Return the *smallest positive integer* that is *not present* in `nums`.

You must implement an algorithm that runs in  $O(n)$  time and uses  $O(1)$  auxiliary space.

### Example 1:

**Input:** `nums = [1,2,0]` **Output:** 3 **Explanation:** The numbers in the range [1,2] are all in the array.

### Example 2:

**Input:** `nums = [3,4,-1,1]` **Output:** 2 **Explanation:** 1 is in the array but 2 is missing.

### Example 3:

**Input:** `nums = [7,8,9,11,12]` **Output:** 1 **Explanation:** The smallest positive integer 1 is missing.

### Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

```
class Solution { public: int firstMissingPositive(vector& nums) { } };
```

Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

- 1. Each of the digits 1-9 must occur exactly once in each row.
- 2. Each of the digits 1-9 must occur exactly once in each column.
- 3. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

The '.' character indicates empty cells.

Example 1:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
Input: board =
[["5","3",".",".","7",".",".","."],["6",".",".","1","9","5",".","."],[".","9","8",".",".",".","6","."],["8","."."5","3","4","2","5","6","7"],["1","9","8","3","4","2","5","6","7"],["8","5"]
Output:
[["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],["1","9","8","3","4","2","5","6","7"],["8","5","3","4","2","5","6","7"],["1","9","8","3","4","2","5","6","7"],["8","5"]
```

Explanation: The input board is shown above and the only valid solution is shown below:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Constraints:

- `board.length == 9`
  - `board[i].length == 9`
  - `board[i][j]` is a digit or '.'.
  - It is **guaranteed** that the input board has only one solution.
- ```
class Solution { public: void solveSudoku(vector<vector<char>>& board) { } };
```

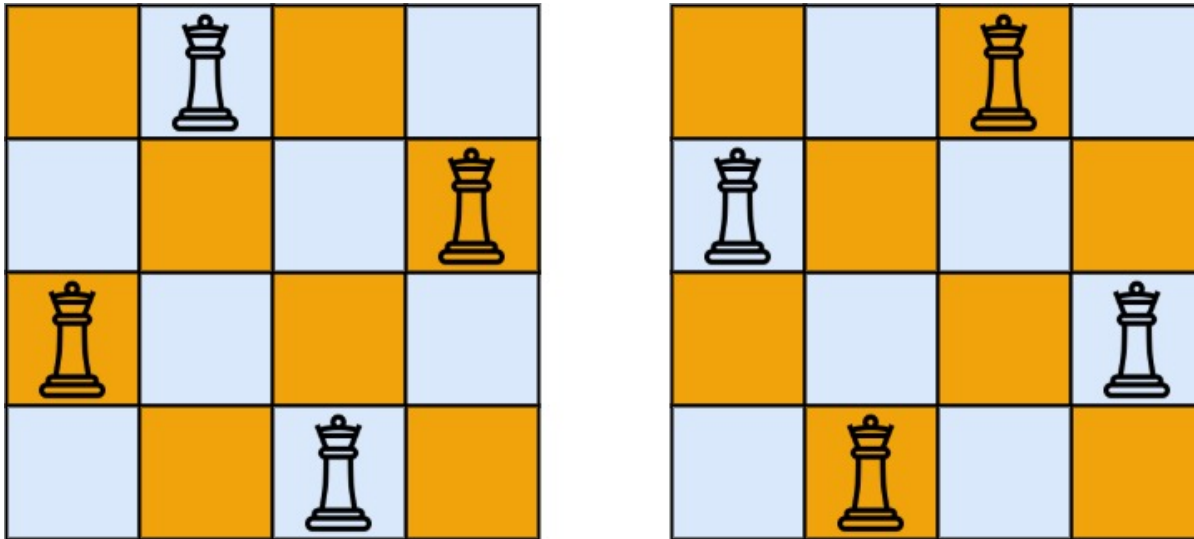
## N-Queens

The **n-queens** puzzle is the problem of placing  $n$  queens on an  $n \times n$  chessboard such that no two queens attack each other.

Given an integer  $n$ , return *all distinct solutions to the n-queens puzzle*. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

### Example 1:



**Input:**  $n = 4$  **Output:** `[[".Q..", "...Q", "Q...", "..Q."], ["..Q.", "Q...", "...Q", ".Q.."]]` **Explanation:** There exist two distinct solutions to the 4-queens puzzle as shown above

### Example 2:

**Input:**  $n = 1$  **Output:** `[["Q"]]`

### Constraints:

- $1 \leq n \leq 9$

```
class Solution { public: vector<vector<string>> solveNQueens(int n) {} };
```

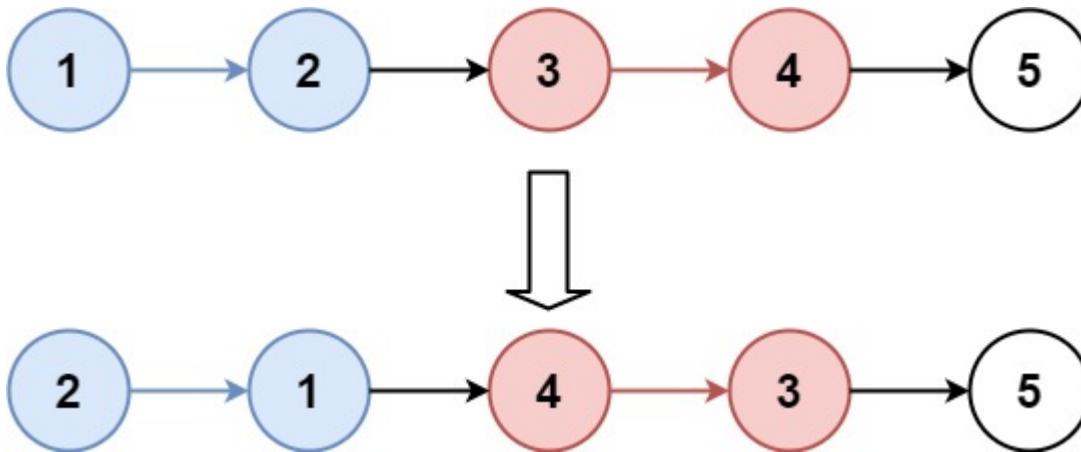
## Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list  $k$  at a time, and return *the modified list*.

$k$  is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of  $k$  then left-out nodes, in the end, should remain as it is.

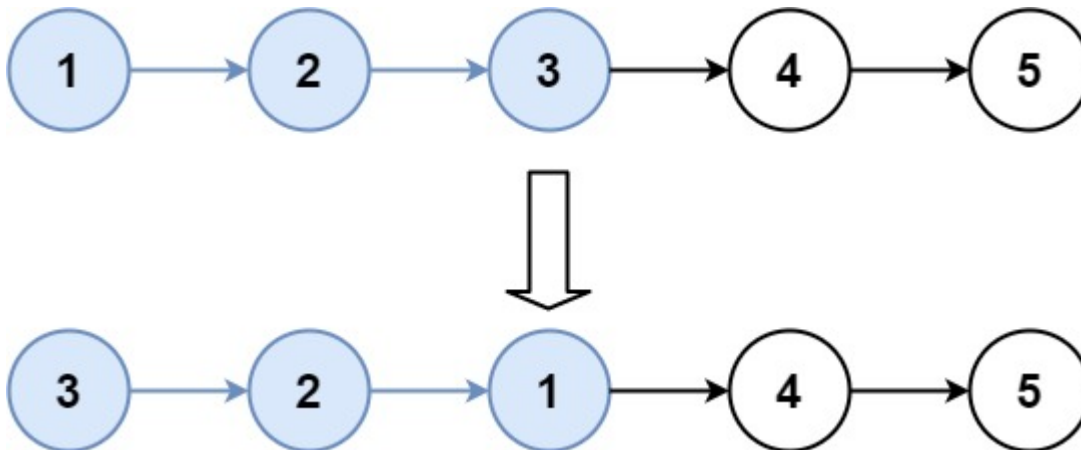
You may not alter the values in the list's nodes, only nodes themselves may be changed.

**Example 1:**



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

**Example 2:**



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

**Constraints:**

- The number of nodes in the list is  $n$ .
- $1 \leq k \leq n \leq 5000$
- $0 \leq \text{Node.val} \leq 1000$

**Follow-up:** Can you solve the problem in  $O(1)$  extra memory space?



```
/** * 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* reverseKGroup(ListNode* head, int k) { }  
};
```

## Merge k Sorted Lists

You are given an array of  $k$  linked-lists `lists`, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

### Example 1:

**Input:** `lists = [[1,4,5],[1,3,4],[2,6]]` **Output:** `[1,1,2,3,4,4,5,6]` **Explanation:** The linked-lists are: [ 1->4->5, 1->3->4, 2->6 ] merging them into one sorted list: 1->1->2->3->4->4->5->6

### Example 2:

**Input:** `lists = []` **Output:** `[]`

### Example 3:

**Input:** `lists = [[]]` **Output:** `[]`

### Constraints:

- $k == \text{lists.length}$
- $0 \leq k \leq 10^4$
- $0 \leq \text{lists}[i].\text{length} \leq 500$
- $-10^4 \leq \text{lists}[i][j] \leq 10^4$
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed  $10^4$ .

```
/** * 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* mergeKLists(vector< ListNode*> lists) { } };
```