1. Remove Element

**Given an integer array nums and an integer val, remove all occurrences of val in nums** [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)**. The relative order of the elements may be changed.**

**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**](https://en.wikipedia.org/wiki/In-place_algorithm) **with O(1) extra memory.**

**Custom Judge:**

**The judge will test your solution with the following code:**

**int[] nums = [...]; // Input array**

**int val = ...; // Value to remove**

**int[] expectedNums = [...]; // The expected answer with correct length.**

**// It is sorted with no values equaling val.**

**int k = removeElement(nums, val); // Calls your implementation**

**assert k == expectedNums.length;**

**sort(nums, 0, k); // Sort the first k elements of nums**

**for (int i = 0; i < actualLength; i++) {**

**assert nums[i] == expectedNums[i];**

**}**

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

**Example 1:**

**Input: nums = [3,2,2,3], val = 3**

**Output: 2, nums = [2,2,\_,\_]**

**Explanation: Your function should return k = 2, with the first two elements of nums being 2.**

**It does not matter what you leave beyond the returned k (hence they are underscores).**

**Example 2:**

**Input: nums = [0,1,2,2,3,0,4,2], val = 2**

**Output: 5, nums = [0,1,4,0,3,\_,\_,\_]**

**Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.**

**Note that the five elements can be returned in any order.**

**It does not matter what you leave beyond the returned k (hence they are underscores).**

**Constraints:**

* **0 <= nums.length <= 100**
* **0 <= nums[i] <= 50**
* **0 <= val <= 100**

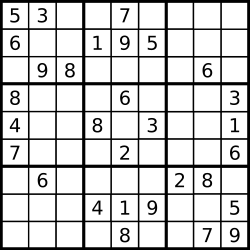
**Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:**

1. **Each row must contain the digits 1-9 without repetition.**
2. **Each column must contain the digits 1-9 without repetition.**
3. **Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.**

**Note:**

* **A Sudoku board (partially filled) could be valid but is not necessarily solvable.**
* **Only the filled cells need to be validated according to the mentioned rules.**

**Example 1:**

****

**Input: board =**

**[["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"]]**

**Output: true**

**Example 2:**

**Input: board =**

**[["8","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"]]**

**Output: false**

**Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.**

**Constraints:**

* **board.length == 9**
* **board[i].length == 9**
* **board[i][j] is a digit 1-9 or '.'.**

37. Sudoku Solver

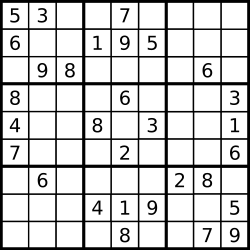
**Write a program to solve a Sudoku puzzle by filling the empty cells.**

**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:**

****

**Input: board = [["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"]]**

**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","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"]]**

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

****

**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.**

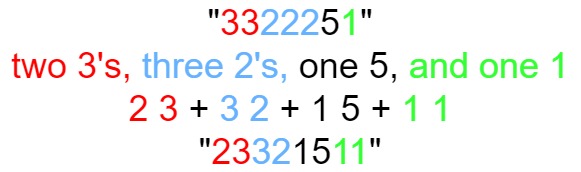
3.Count and Say

**The count-and-say sequence is a sequence of digit strings defined by the recursive formula:**

* **countAndSay(1) = "1"**
* **countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.**

**To determine how you "say" a digit string, split it into the minimal number of substrings such that each substring contains exactly one unique digit. Then for each substring, say the number of digits, then say the digit. Finally, concatenate every said digit.**

**For example, the saying and conversion for digit string "3322251":**

****

**Given a positive integer n, return *the* nth *term of the count-and-say sequence*.**

**Example 1:**

**Input: n = 1**

**Output: "1"**

**Explanation: This is the base case.**

**Example 2:**

**Input: n = 4**

**Output: "1211"**

**Explanation:**

**countAndSay(1) = "1"**

**countAndSay(2) = say "1" = one 1 = "11"**

**countAndSay(3) = say "11" = two 1's = "21"**

**countAndSay(4) = say "21" = one 2 + one 1 = "12" + "11" = "1211"**

**Constraints:**

* **1 <= n <= 30**

39. Combination Sum

**Given an array of distinct integers candidates and a target integer target, return *a list of all unique combinations of* candidates *where the chosen numbers sum to* target*.* You may return the combinations in any order.**

**The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.**

**The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.**

**Example 1:**

**Input: candidates = [2,3,6,7], target = 7**

**Output: [[2,2,3],[7]]**

**Explanation:**

**2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.**

**7 is a candidate, and 7 = 7.**

**These are the only two combinations.**

**Example 2:**

**Input: candidates = [2,3,5], target = 8**

**Output: [[2,2,2,2],[2,3,3],[3,5]]**

**Example 3:**

**Input: candidates = [2], target = 1**

**Output: []**

**Constraints:**

* **1 <= candidates.length <= 30**
* **2 <= candidates[i] <= 40**
* **All elements of candidates are distinct.**
* **1 <= target <= 40**

40. Combination Sum II

**Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.**

**Each number in candidates may only be used once in the combination.**

**Note: The solution set must not contain duplicate combinations.**

**Example 1:**

**Input: candidates = [10,1,2,7,6,1,5], target = 8**

**Output:**

**[**

**[1,1,6],**

**[1,2,5],**

**[1,7],**

**[2,6]**

**]**

**Example 2:**

**Input: candidates = [2,5,2,1,2], target = 5**

**Output:**

**[**

**[1,2,2],**

**[5]**

**]**

**Constraints:**

* **1 <= candidates.length <= 100**
* **1 <= candidates[i] <= 50**
* **1 <= target <= 30**

Permutations II

**Given a collection of numbers, nums, that might contain duplicates, return *all possible unique permutations in any order.***

**Example 1:**

**Input: nums = [1,1,2]**

**Output:**

**[[1,1,2],**

**[1,2,1],**

**[2,1,1]]**

**Example 2:**

**Input: nums = [1,2,3]**

**Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]**

**Constraints:**

* **1 <= nums.length <= 8**
* **-10 <= nums[i] <= 10**

53. Maximum Subarray

**Given an integer array nums, find the subarray which has the largest sum and return *its sum*.**

**Example 1:**

**Input: nums = [-2,1,-3,4,-1,2,1,-5,4]**

**Output: 6**

**Explanation: [4,-1,2,1] has the largest sum = 6.**

**Example 2:**

**Input: nums = [1]**

**Output: 1**

**Example 3:**

**Input: nums = [5,4,-1,7,8]**

**Output: 23**

**Constraints:**

* **1 <= nums.length <= 105**
* **-104 <= nums[i] <= 104**

Length of Last Word

**Given a string s consisting of words and spaces, return *the length of the last word in the string.***

**A word is a maximal substring consisting of non-space characters only.**

**Example 1:**

**Input: s = "Hello World"**

**Output: 5**

**Explanation: The last word is "World" with length 5.**

**Example 2:**

**Input: s = " fly me to the moon "**

**Output: 4**

**Explanation: The last word is "moon" with length 4.**

**Example 3:**

**Input: s = "luffy is still joyboy"**

**Output: 6**

**Explanation: The last word is "joyboy" with length 6.**

**Constraints:**

* **1 <= s.length <= 104**
* **s consists of only English letters and spaces ' '.**
* **There will be at least one word in s.**

Permutation Sequence

**The set [1, 2, 3, ..., n] contains a total of n! unique permutations.**

**By listing and labeling all of the permutations in order, we get the following sequence for n = 3:**

1. **"123"**
2. **"132"**
3. **"213"**
4. **"231"**
5. **"312"**
6. **"321"**

**Given n and k, return the kth permutation sequence.**

**Example 1:**

**Input: n = 3, k = 3**

**Output: "213"**

**Example 2:**

**Input: n = 4, k = 9**

**Output: "2314"**

**Example 3:**

**Input: n = 3, k = 1**

**Output: "123"**

**Constraints:**

* **1 <= n <= 9**
* **1 <= k <= n!**