

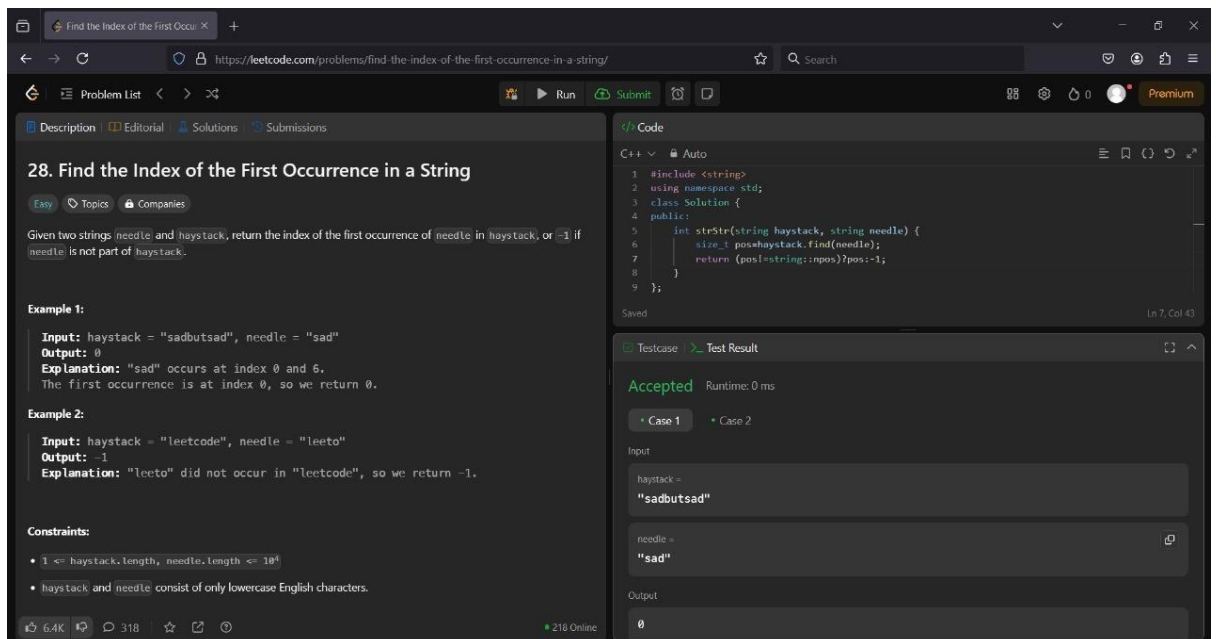
# DAA HOLIDAY ASSIGNMENT

P.SAI PRIYA  
2211CS020404

AIML-THETA

1) find-the-index-of-the-first-occurrence-in-a-string

<https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/description/>



The screenshot shows the LeetCode interface for the problem "28. Find the Index of the First Occurrence in a String". The problem is categorized as "Easy". The description states: "Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`." Two examples are provided: Example 1 with `haystack = "sadbutsad"` and `needle = "sad"` resulting in `0`; and Example 2 with `haystack = "leetcode"` and `needle = "leeto"` resulting in `-1`. Constraints include that both strings have lengths up to  $10^4$  and consist of lowercase English characters. The C++ code on the right uses `std::string::find` to solve the problem. The test result shows "Accepted" with a runtime of 0 ms.

```
1 #include <string>
2 using namespace std;
3 class Solution {
4 public:
5     int strStr(string haystack, string needle) {
6         size_t pos=haystack.find(needle);
7         return (pos!=string::npos)?pos:-1;
8     };
9 }
```

2) Bitwise AND of numbers range

<https://leetcode.com/problems/bitwise-and-of-numbers-range/>

**201. Bitwise AND of Numbers Range**

Given two integers `left` and `right` that represent the range `[left, right]`, return the *bitwise AND* of all numbers in this range, inclusive.

**Example 1:**  
Input: `left = 5, right = 7`  
Output: `4`

**Example 2:**  
Input: `left = 0, right = 0`  
Output: `0`

**Example 3:**  
Input: `left = 1, right = 2147483647`  
Output: `0`

**Constraints:**

- $0 \leq \text{left} \leq \text{right} \leq 2^{31} - 1$

```
class Solution {
public:
    int rangeBitwiseAnd(int left, int right) {
        int shift=0;
        while(left < right){
            left>>=1;
            right>>=1;
            shift++;
        }
        return left<<shift;
    }
};
```

Testcase: **Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

Input

left = 5

right = 7

### 3)SQRT(X)

<https://leetcode.com/problems/sqrtx/>

**69. Sqrt(x)**

Given a non-negative integer `x`, return the *square root of `x`* rounded down to the nearest integer. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

**Example 1:**  
Input: `x = 4`  
Output: `2`  
Explanation: The square root of 4 is 2, so we return 2.

**Example 2:**  
Input: `x = 8`  
Output: `2`  
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

**Constraints:**

```
class Solution:
    def mySqrt(self, x):
        if x < 2:
            return x
        low, high = 0, x // 2 + 1
        result = 0
        while low <= high:
            mid = (low + high) // 2
            if mid * mid == x:
                return mid
            elif mid * mid < x:
                result = mid
                low = mid + 1
            else:
                high = mid - 1
        return result
```

Testcase: **Accepted** Runtime: 0 ms

Case 1 Case 2

Input

x = 4

### 4)Largest Number

<https://leetcode.com/problems/largest-number/description/>

The screenshot shows the LeetCode problem page for "179. Largest Number". The problem description states: "Given a list of non-negative integers `nums`, arrange them such that they form the largest number and return it. Since the result may be very large, so you need to return a string instead of an integer." Example 1 shows input `nums = [10, 2]` and output `"210"`. Example 2 shows input `nums = [3, 30, 34, 5, 9]` and output `"9534330"`. Constraints include `1 <= nums.length <= 100` and `0 <= nums[i] <= 109`. The code editor on the right shows a Python solution using a custom comparator to sort the numbers. The test results show the solution is "Accepted" with a runtime of 0 ms for Case 1.

179. Largest Number

Medium Topics Companies

Given a list of non-negative integers `nums`, arrange them such that they form the largest number and return it. Since the result may be very large, so you need to return a string instead of an integer.

Example 1:

Input: `nums = [10, 2]`  
Output: `"210"`

Example 2:

Input: `nums = [3, 30, 34, 5, 9]`  
Output: `"9534330"`

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 109`

9K 186 44 Online

```
1 from functools import cmp_to_key
2 class Solution:
3     def largestNumber(self, nums):
4         nums = list(map(str, nums))
5         def compare(x, y):
6             if x + y > y + x:
7                 return -1
8             elif x + y < y + x:
9                 return 1
10            else:
11                return 0
12        nums.sort(key=cmp_to_key(compare))
13        result = ''.join(nums)
14        return '0' if result[0] == '0' else result
```

Saved Ln 14, Col 51

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`nums =`  
`[10, 2]`

Output

## 5)Valid Parenthesis

<https://leetcode.com/problems/valid-parentheses/description/>

The screenshot shows the LeetCode problem page for "20. Valid Parentheses". The problem description states: "Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid. An input string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type." Example 1 shows input `s = "()"` and output `true`. Example 2 shows input `s = "() [] {}"` and output `true`. Example 3 shows input `s = "( [ ] { )"` and output `true`. The code editor on the right shows a Python solution using a stack to validate the parentheses. The test results show the solution is "Accepted" with a runtime of 0 ms for Case 1.

20. Valid Parentheses

Easy Topics Companies Hint

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid. An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.
- Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: `s = "()"`  
Output: `true`

Example 2:

Input: `s = "() [] {}"`  
Output: `true`

Example 3:

Input: `s = "( [ ] { )"`  
Output: `true`

25K 457 532 Online

```
1 class Solution:
2     def isValid(self, s):
3         stack = []
4         bracket_map = {'(': ')', '(': ')', '(': ')', '(': ')', '(': ')'}
5         for char in s:
6             if char in bracket_map:
7                 top_element = stack.pop() if stack else '#'
8                 if bracket_map[char] != top_element:
9                     return False
10            else:
11                stack.append(char)
12        return not stack
```

Saved Ln 12, Col 27

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4

Input

`s =`  
`"()"`

Output

`true`

## 6)Merge Two Sorted Lists

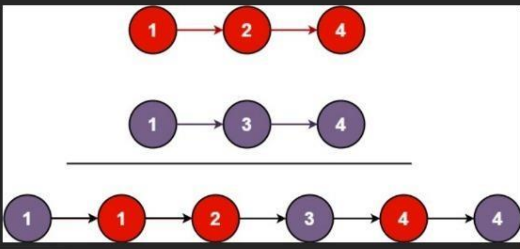
<https://leetcode.com/problems/merge-two-sorted-lists/description/>

21. Merge Two Sorted Lists

Easy Topics Companies

You are given the heads of two sorted linked lists `list1` and `list2`. Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

Example 1:



```

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def mergeTwoLists(self, list1, list2):
        dummy = ListNode()
        current = dummy
        while list1 and list2:
            if list1.val < list2.val:
                current.next = list1
                list1 = list1.next
            else:
                current.next = list2
                list2 = list2.next
            current = current.next
        if list1:
            current.next = list1
        elif list2:
            current.next = list2
        return dummy.next

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

## 7) Remove Duplicates from sorted list

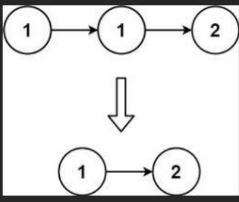
<https://leetcode.com/problems/remove-duplicates-from-sortedlist/description/>

83. Remove Duplicates from Sorted List

Easy Topics Companies


Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list **sorted** as well.

Example 1:



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

Example 2:



```

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def deleteDuplicates(self, head):
        current = head
        while current and current.next:
            if current.val == current.next.val:
                # Skip the next node if it's a duplicate
                current.next = current.next.next
            else:
                current = current.next
        return head

```

Accepted Runtime: 0 ms

Case 1 Case 2

Input: head = [1,1,2]

## 8) Find peak Element

<https://leetcode.com/problems/find-peak-element/>

Find Peak Element - LeetCode

162. Find Peak Element

A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in  $O(\log n)$  time.

**Example 1:**

Input: `nums = [1,2,3,1]`  
Output: `2`  
Explanation: 3 is a peak element and your function should return the index number 2.

**Example 2:**

Input: `nums = [1,2,1,3,5,6,4]`  
Output: `5`  
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Python

```

class Solution:
    def findPeakElement(self, nums):
        left, right = 0, len(nums) - 1
        while left < right:
            mid = (left + right) // 2
            if nums[mid] < nums[mid + 1]:
                left = mid + 1
            else:
                right = mid
        return left

```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`nums =`  
`[1,2,3,1]`

Output

`2`

Expected

## 9) Binary Tree Inorder Traversal

<https://leetcode.com/problems/binary-tree-inorder-traversal/>

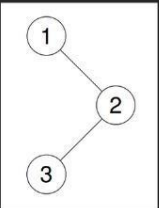
Binary Tree Inorder Traversal - L X

94. Binary Tree Inorder Traversal

Given the `root` of a binary tree, return the *inorder traversal* of its nodes' values.

**Example 1:**

Input: `root = [1,null,2,3]`  
Output: `[1,3,2]`  
Explanation:



```

graph TD
    1((1)) --> 3L((3))
    1 --> 2((2))
    2 --> 3R((3))

```

Python

```

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def inorderTraversal(self, root):
        result = []
        def inorder(node):
            if node:
                inorder(node.left)
                result.append(node.val)
                inorder(node.right)
        inorder(root)
        return result

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4

Input

`root =`  
`[1,null,2,3]`

## 10) N-Queens

<https://leetcode.com/problems/n-queens/>





## DAA case-study

### Scenario:

An e-commerce platform is implementing a feature where products need to be sorted by various attributes (e.g. price, rating and name). The product list contains millions of items, and the sorting operation needs to be efficient and scalable.

1. What are the time and space complexities of the commonly used sorting algorithms (Quick sort, Merge Sort)?
  2. How do the characteristics of the data (e.g. range of prices, product name lengths) impact the choice of sorting algorithm?
- 1) Time and space complexities of common sorting Algorithms.

### Quick sort:

Time complexity (Best):  $O(n \log n)$ .

Time complexity (Average):  $O(n \log n)$

Time complexity (worst):  $O(n^2)$  (unbalanced pivot)

Space Complexity:  $O(\log n)$  auxiliary

### Merge Sort:

Time Complexity (Best):  $O(n \log n)$

Time Complexity (Average):  $O(n \log n)$

Time complexity (worst):  $O(n \log n)$

Space Complexity:  $O(n)$  auxiliary.

## 2) Impact Of Data characteristics on Sorting Algorithm choice

### 1. Range of prices or Numerical Data:

- For a small range of values (e.g., product prices), counting sort or radix sort may be more efficient than comparison-based algorithms, achieving linear time complexity  $O(n+k)$ .
- For a large or arbitrary range, comparison-based algorithms like Quick Sort or Merge Sort are preferred.

### 2. Product Name Lengths (String Sorting):

- Lexicographic sorting involves comparing strings character by character. Algorithms like Quick Sort and Merge Sort still perform well but may be slower due to increased comparison times.
- Radix Sort can be used for fixed-length strings or cases where the length is bounded and known, providing near-linear time complexity.

### 3. Distribution and size of Data:

- Uniformly distributed data: Quick Sort performs well with randomized pivot selection strategies.
- Nearly Sorted Data: Inserting Sort (or) Trim Sort may outperform others due to better handling of such input.

### 4. Stability Requirements:

- Merge Sort is stable, meaning equal elements retain their relative order. Quick Sort is generally not stable unless modified.

### 5. Memory Constraints:

- If memory usage is a concern, Quick Sort (with-in place sorting) is often more suitable than Merge Sort.