

1.) Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`. You may assume that each input would have exactly one solution, and you may not use the same element twice.

Sent by you: Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`. You may assume that each input would have exactly one solution, and you may not use the same element twice.

Input: `nums = [2,7,11,15]`, `target = 9` Output: `[0,1]` Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Sent by you: Input: `nums = [2,7,11,15]`, `target = 9` Output: `[0,1]` Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

The screenshot displays a LeetCode submission page for the 'Two Sum' problem. The submission is marked as 'Accepted' and was submitted by 'Surya Narayanan' on June 03, 2024, at 22:52. The performance metrics show a runtime of 57 ms, which beats 71.00% of users with Python3, and a memory usage of 17.67 MB, which beats 54.93% of users with Python3. A horizontal bar chart illustrates the runtime distribution, with the user's solution at 57 ms. The code editor shows a Python3 solution using a hash map (preMap) to store the difference between the target and the current element. The test case input is `nums = [2,7,11,15]` and `target = 9`, with the output being `[0,1]`.

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        preMap = {}
        for i, n in enumerate(nums):
            diff = target - n
            if diff in preMap:
                return [preMap[diff], i]
            preMap[n] = i
```

Time : $O(n)$

Space: $O(n)$

2.) **Add Two Numbers** You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1: Input: l1 = [2,4,3], l2 = [5,6,4] Output: [7,0,8] Explanation: 342 + 465 = 807.

The screenshot displays the LeetCode submission interface for the "Add Two Numbers" problem. The top section shows the problem description and a code editor with a Python solution. The bottom section shows the test result, indicating that the solution was accepted.

Code:

```
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummyHead = ListNode(0)
        tail = dummyHead
        carry = 0

        while l1 is not None or l2 is not None or carry != 0:
            d1 = l1.val if l1 is not None else 0
            d2 = l2.val if l2 is not None else 0

            sum = d1 + d2 + carry
            digit = sum % 10
            carry = sum // 10

            newNode = ListNode(digit)
            tail.next = newNode
            tail = tail.next

            l1 = l1.next if l1 is not None else None
            l2 = l2.next if l2 is not None else None

        result = dummyHead.next
        dummyHead.next = None
        return result
```

Test Result:

Accepted Runtime: 46 ms

Case 1 Case 2 Case 3

Input

l1 = [2, 4, 3]

l2 = [5, 6, 4]

Output

[7, 0, 8]

Expected

[7, 0, 8]

Time : $O(\max(m,n))$

3. **Longest Substring without Repeating Characters** Given a string *s*, find the length of the longest substring without repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3 Explanation: The answer is "abc", with the length of 3.

Example 2: Input: s = "bbbbbb" Output: 1 Explanation: The answer is "b", with the length of 1

The screenshot shows a LeetCode submission page for the problem "Length of Longest Substring". The submission is marked as "Accepted" and was submitted by "Surya Narayanan" on June 03, 2024, at 23:45. The runtime is 49 ms, which beats 81.40% of users with Python3. The memory usage is 16.51 MB, which beats 92.41% of users with Python3. A bar chart shows the submission's performance relative to other users. The code is a Python class named "Solution" with a method "lengthOfLongestSubstring" that uses a sliding window approach with a set to track characters.

```
1 class Solution:
2     def lengthOfLongestSubstring(self, s: str) -> int:
3         charset = set()
4         l = 0
5         res = 0
6         for r in range(len(s)):
7             while s[r] in charset:
8                 charset.remove(s[l])
9                 l += 1
10            charset.add(s[r])
11            res = max(res, r - l + 1)
12        return res
13
```

The screenshot shows the test case results for the "Length of Longest Substring" problem. The submission is marked as "Accepted" with a runtime of 40 ms. There are three test cases: Case 1, Case 2, and Case 3. Case 1 is selected, showing the input s = "abcabcbb" and the output 3.

Accepted Runtime: 40 ms

- Case 1
- Case 2
- Case 3

Input

s = "abcabcbb"

Output

3

Time: $O(n)$

Space: $O(n)$

4. Median of Two Sorted Arrays Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be $O(\log(m+n))$.

Example 1: Input: nums1 = [1,3], nums2 = [2] Output: 2.00000 Explanation: merged array = [1,2,3] and median is 2.

Example 2: Input: nums1 = [1,2], nums2 = [3,4] Output: 2.50000 Explanation: merged array = [1,2,3,4] and median is $(2 + 3) / 2 = 2.5$

Accepted

Surya Narayanan submitted at Jun 04, 2024 00:43

Runtime

80 ms

Beats 60.73% of users with Python3

Memory

16.88 MB

Beats 76.30% of users with Python3

Python3 Auto

```
1 class Solution:
2     def findMedianSortedArrays(self, nums1, nums2):
3         n = len(nums1)
4         m = len(nums2)
5         i = 0
6         j = 0
7         m1 = 0
8         m2 = 0
9         for count in range(0, (n + m) // 2 + 1):
10             m2 = m1
11             if i < n and j < m:
12                 if nums1[i] > nums2[j]:
13                     m1 = nums2[j]
14                     j += 1
15                 else:
16                     m1 = nums1[i]
17                     i += 1
18             elif i < n:
19                 m1 = nums1[i]
20                 i += 1
21             else:
22                 m1 = nums2[j]
23                 j += 1
```

Saved

Ln 8, Col 15

Testcase Test Result

Python3 Auto

```
21         else:
22             m1 = nums2[j]
23             j += 1
24         if (n + m) % 2 == 1:
25             return float(m1)
26         else:
27             ans = float(m1) + float(m2)
28             return ans / 2.0
```

Saved

Ln 8, Col 15

Testcase Test Result

Input

nums1 =

[1, 3]

nums2 =

[2]

Output

2.00000

5.) **Longest Palindromic Substring** Given a string s , return the longest palindromic substring in s .

Example 1: Input: $s = \text{"babad"}$ Output: "bab" Explanation: "aba" is also a valid answer.

Example 2: Input: $s = \text{"cbbd"}$ Output: "bb"

The screenshot shows a LeetCode submission interface. On the left, the submission is marked as 'Accepted' by user 'Surya Narayanan' on June 04, 2024. Performance metrics show a runtime of 280 ms (beating 79.53% of users) and memory usage of 16.46 MB (beating 96.63% of users). A bar chart visualizes the user's performance against other submissions. On the right, the Python code is displayed, implementing a center expansion algorithm. The code defines a 'Solution' class with a 'longestPalindrome' method that iterates through each character in the string and expands outwards to find the longest palindrome.

```
1 class Solution:
2     def longestPalindrome(self, s: str) -> str:
3         if len(s) <= 1:
4             return s
5         def expand_from_center(left, right):
6             while left >= 0 and right < len(s) and s[left] == s[right]:
7                 left -= 1
8                 right += 1
9             return s[left + 1:right]
10        max_str = s[0]
11        for i in range(len(s) - 1):
12            odd = expand_from_center(i, i)
13            even = expand_from_center(i, i + 1)
14
15            if len(odd) > len(max_str):
16                max_str = odd
17            if len(even) > len(max_str):
18                max_str = even
19        return max_str
```

The screenshot shows the input and output for the 'Longest Palindromic Substring' problem. The input string s is "babad" and the output is "bab" .

Time: $O(n^2)$

Space: $O(1)$

6. Zigzag Conversion The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion given a number of rows: string convert(string s, int numRows);

Example 1: Input: s = "PAYPALISHIRING", numRows = 3 Output: "PAHNAPLSIIGYIR"

Example 2: Input: s = "PAYPALISHIRING", numRows = 4 Output: "PINALSIGYAHRPI"

The screenshot shows a LeetCode submission page for the Zigzag Conversion problem. On the left, the submission status is 'Accepted' by Surya Narayanan at Jun 04, 2024 00:58. Performance metrics show a runtime of 48 ms, beating 74.11% of users with Python3, and a memory usage of 16.73 MB, beating 36.75% of users with Python3. A bar chart shows the submission's performance relative to others. On the right, the Python code is displayed:

```
def convert(self, s: str, numRows: int) -> str:
    if numRows == 1 or numRows >= len(s):
        return s

    idx, d = 0, 1
    rows = [[] for _ in range(numRows)]

    for char in s:
        rows[idx].append(char)
        if idx == 0:
            d = 1
        elif idx == numRows - 1:
            d = -1
        idx += d

    for i in range(numRows):
        rows[i] = ''.join(rows[i])

    return ''.join(rows)
```

Input

s =
"PAYPALISHIRING"

numRows =
3

Output

"PAHNAPLSIIGYIR"

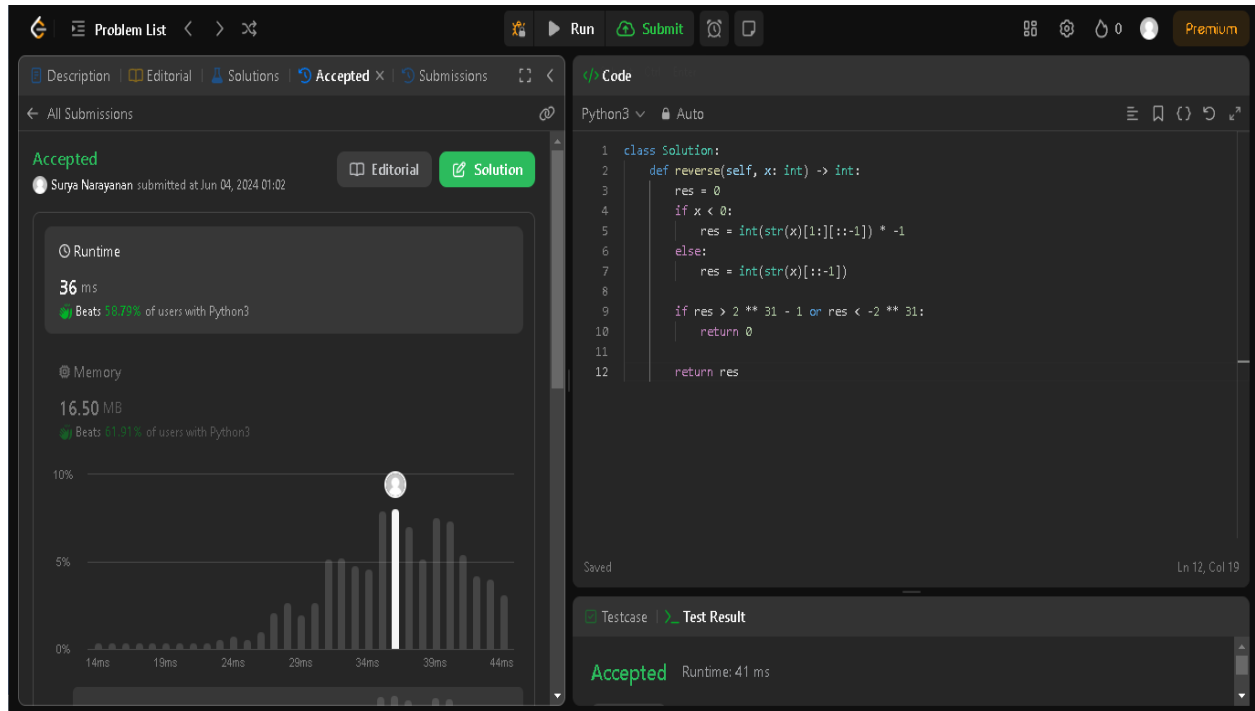
Time: $O(n)$

Space: $O(n)$

7. Reverse Integer Given a signed 32-bit integer x , return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range $[-2^{31}, 2^{31} - 1]$, then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1: Input: $x = 123$ Output: 321

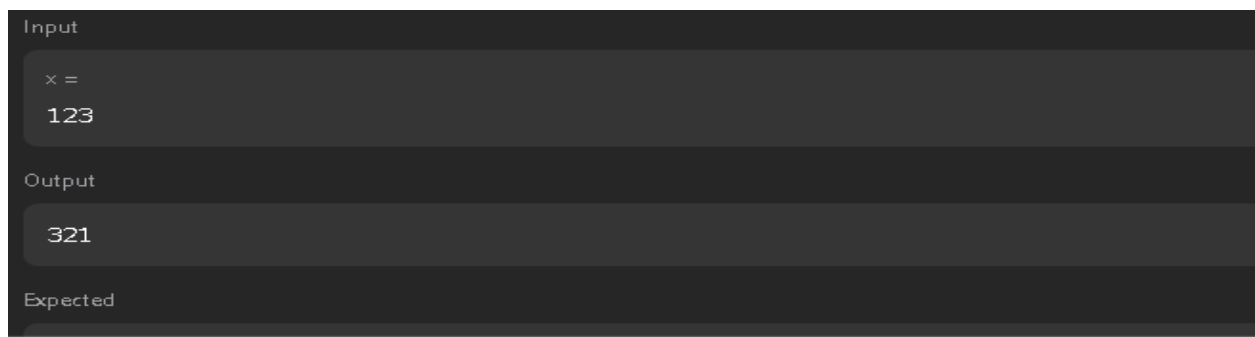
Example 2: Input: $x = -123$ Output: -321



The screenshot shows a LeetCode interface with the following details:

- Problem Status:** Accepted. Submitted by Surya Narayanan on Jun 04, 2024 at 01:02.
- Runtime:** 36 ms, Beats 98.73% of users with Python3.
- Memory:** 16.50 MB, Beats 81.81% of users with Python3.
- Performance Graph:** A bar chart showing runtime performance across various test cases, with a peak around 34ms.
- Code:** Python3 solution using string reversal and range checking.

```
1 class Solution:
2     def reverse(self, x: int) -> int:
3         res = 0
4         if x < 0:
5             res = int(str(x)[1:][::-1]) * -1
6         else:
7             res = int(str(x)[::-1])
8
9         if res > 2 ** 31 - 1 or res < -2 ** 31:
10             return 0
11
12         return res
```
- Testcase:** Accepted, Runtime: 41 ms.



The input and output fields are as follows:

Input	Output
$x = 123$	321

Expected: (The expected output field is empty in the image)

Time: $O(n)$

Space: $O(n)$

8. String to Integer (atoi) Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function). The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31} , and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result

The screenshot displays the LeetCode submission interface for the "String to Integer (atoi)" problem. The left sidebar shows the problem status as "Accepted" and the submission details: "Surya Narayanan submitted at Jun 04, 2024 01:07". Below this, the runtime is 32 ms, which beats 89.02% of users with Python3. The memory usage is 16.30 MB, which beats 99.64% of users with Python3. A bar chart shows the distribution of runtime times, with the user's submission at 32 ms. The main area displays the Python code for the solution:

```
1 class Solution(object):
2     def myAtoi(self, s):
3         num = '0123456789'
4         res = ''
5         for x in s:
6             if x == ' ' and len(res) == 0:
7                 continue
8             if x != ' ' and (x in '+' or x in '-' or x in num) and len(res) == 0:
9                 res += x
10            elif x in num:
11                res += x
12            else:
13                break
14        if res == '' or res in '+-':
15            return 0
16        else:
17            if int(res) < -(2**31):
18                return -(2**31)
19            elif int(res) > (2**31 - 1):
20                return (2**31 - 1)
21            else:
22                return int(res)
```

The bottom of the interface shows tabs for "Testcase" and "Test Result".

Input

`s =`
`"42"`

Output

42

Time : $O(n)$

Space : $O(1)$

9.) Given an integer x , return true if x is a palindrome, and false otherwise.

Example 1: Input: $x = 121$ Output: true Explanation: 121 reads as 121 from left to right and from right to left.

Example 2: Input: $x = -121$ Output: false Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

The screenshot shows a LeetCode problem solution for "Is Palindrome". The problem is marked as "Accepted". The user "Surya Narayanan" submitted the solution on Jun 04, 2024, at 01:12. The solution is written in Python3 and is 54 ms, which beats 53.08% of users with Python3. The memory usage is 16.58 MB, which beats 58.41% of users with Python3. A bar chart shows the runtime distribution, with the user's solution at 54 ms. The code is as follows:

```
1 class Solution:
2     def isPalindrome(self, x: int) -> bool:
3         if x < 0:
4             return False
5         reversed_num = 0
6         temp = x
7         while temp != 0:
8             digit = temp % 10
9             reversed_num = reversed_num * 10 + digit
10            temp //= 10
11        return reversed_num == x
```

The test result shows the input $x = 121$ and the output `true`.

10.) **Regular Expression Matching** Given an input string s and a pattern p , implement regular expression matching with support for $'.'$ and $'*'$ where:

- $'.'$ Matches any single character.
- $'*'$ Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

Example 1: Input: $s = "aa"$, $p = "a"$ Output: false Explanation: "a" does not match the entire string "aa".

The image displays two screenshots of a LeetCode submission interface for the "Regular Expression Matching" problem. The interface is in dark mode and shows the problem description, submission status, and a code editor.

Top Screenshot:

- Problem List:** Shows the problem name "Regular Expression Matching" and the submission status "Accepted".
- Submission Details:** The submission was made by "Surya Narayanan" on "Jun 04, 2024 01:15". It shows a runtime of "40 ms" and memory usage of "16.60 MB", both of which beat 90.01% of users with Python3.
- Code Editor:** The code is written in Python3 and uses a class-based solution with a backtracking function. The code is as follows:

```
1 class Solution:
2     def isMatch(self, s: str, p: str) -> bool:
3         i, j = len(s) - 1, len(p) - 1
4         return self.backtrack({}, s, p, i, j)
5
6     def backtrack(self, cache, s, p, i, j):
7         key = (i, j)
8         if key in cache:
9             return cache[key]
10
11         if i == -1 and j == -1:
12             cache[key] = True
13             return True
14
15         if i != -1 and j == -1:
16             cache[key] = False
17             return cache[key]
18
19         if i == -1 and p[j] == '*':
20             k = j
21             while k != -1 and p[k] == '*':
22                 k -= 2
23
```

Bottom Screenshot:

- Problem List:** Shows the problem name "Regular Expression Matching" and the submission status "Accepted".
- Submission Details:** The submission was made by "Surya Narayanan" on "Jun 04, 2024 01:15". It shows a runtime of "40 ms" and memory usage of "16.60 MB", both of which beat 90.01% of users with Python3.
- Code Editor:** The code is written in Python3 and uses a class-based solution with a backtracking function. The code is as follows:

```
22         k -= 2
23
24         if k == -1:
25             cache[key] = True
26             return cache[key]
27
28         cache[key] = False
29         return cache[key]
30
31         if i == -1 and p[j] != '*':
32             cache[key] = False
33             return cache[key]
34
35         if p[j] == '*':
36             if self.backtrack(cache, s, p, i, j - 2):
37                 cache[key] = True
38                 return cache[key]
39
40             if p[j - 1] == s[i] or p[j - 1] == '.':
41                 if self.backtrack(cache, s, p, i - 1, j):
42                     cache[key] = True
43                     return cache[key]
44
45         if p[j] == '.' or s[i] == p[j]:
```

Code

Python3 Auto

43

44

45

46

47

48

49

50

51

return cache[key]

if p[j] == '.' or s[i] == p[j]:

if self.backtrack(cache, s, p, i - 1, j - 1):

cache[key] = True

return cache[key]

cache[key] = False

return cache[key]

SavedLn 51, Col 2

☒ Testcase

☐ Test Result

s =

"aa"

p =

"a"

Output

false

Time : $O(m \cdot n)$

Space: $O(m \cdot n)$