**1)def two\_sum(nums, target):**

**num\_to\_index = {}**

**for i, num in enumerate(nums):**

**complement = target - num**

**if complement in num\_to\_index:**

**return [num\_to\_index[complement], i]**

**num\_to\_index[num] = i**

**nums = [2, 7, 11, 15]**

**target = 9**

**print(two\_sum(nums, target))**

**Output: [0, 1]**

**2) class ListNode:**

**def \_\_init\_\_(self, val=0, next=None):**

**self.val = val**

**self.next = next**

**def add\_two\_numbers(l1, l2):**

**dummy\_head = ListNode(0)**

**current = dummy\_head**

**carry = 0**

**while l1 or l2 or carry:**

**val1 = l1.val if l1 else 0**

**val2 = l2.val if l2 else 0**

**carry, out = divmod(val1 + val2 + carry, 10)**

**current.next = ListNode(out)**

**current = current.next**

**l1 = l1.next if l1 else None**

**l2 = l2.next if l2 else None**

**return dummy\_head.next**

**def create\_linked\_list(lst):**

**dummy\_head = ListNode(0)**

**current = dummy\_head**

**for value in lst:**

**current.next = ListNode(value)**

**current = current.next**

**return dummy\_head.next**

**def print\_linked\_list(node):**

**while node:**

**print(node.val, end=" -> ")**

**node = node.next**

**print("None")**

**l1 = create\_linked\_list([2, 4, 3])**

**l2 = create\_linked\_list([5, 6, 4])**

**result = add\_two\_numbers(l1, l2)**

**print\_linked\_list(result)**

**Output: 7 -> 0 -> 8 -> None**

**3)def length\_of\_longest\_substring(s: str) -> int:**

**char\_index\_map = {}**

**start = 0**

**max\_length = 0**

**for end, char in enumerate(s):**

**if char in char\_index\_map and char\_index\_map[char] >= start:**

**start = char\_index\_map[char] + 1**

**char\_index\_map[char] = end**

**max\_length = max(max\_length, end - start + 1)**

**return max\_length**

**s = "abcabcbb"**

**print(length\_of\_longest\_substring(s))**

**Output: 3**

**4) def find\_median\_sorted\_arrays(nums1, nums2):**

**if len(nums1) > len(nums2):**

**nums1, nums2 = nums2, nums1**

**m, n = len(nums1), len(nums2)**

**imin, imax, half\_len = 0, m, (m + n + 1) // 2**

**while imin <= imax:**

**i = (imin + imax) // 2**

**j = half\_len - i**

**if i < m and nums1[i] < nums2[j-1]:**

**imin = i + 1**

**elif i > 0 and nums1[i-1] > nums2[j]:**

**imax = i - 1**

**else:**

**if i == 0: max\_of\_left = nums2[j-1]**

**elif j == 0: max\_of\_left = nums1[i-1]**

**else: max\_of\_left = max(nums1[i-1], nums2[j-1])**

**if (m + n) % 2 == 1:**

**return max\_of\_left**

**if i == m: min\_of\_right = nums2[j]**

**elif j == n: min\_of\_right = nums1[i]**

**else: min\_of\_right = min(nums1[i], nums2[j])**

**return (max\_of\_left + min\_of\_right) / 2.0**

**nums1 = [1, 3]**

**nums2 = [2]**

**print(find\_median\_sorted\_arrays(nums1, nums2))**

**Output: 2.0**

**5) def convert(s: str, numRows: int) -> str:**

**if numRows == 1 or numRows >= len(s):**

**return s**

**rows = [''] \* numRows**

**index, step = 0, 1**

**for char in s:**

**rows[index] += char**

**if index == 0:**

**step = 1**

**elif index == numRows - 1:**

**step = -1**

**index += step**

**return ''.join(rows)**

**s = "PAYPALISHIRING"**

**numRows = 3**

**print(convert(s, numRows))**

**Output: "PAHNAPLSIIGYIR"**

**6) def convert(s: str, numRows: int) -> str:**

**if numRows == 1 or numRows >= len(s):**

**return s**

**rows = [''] \* numRows**

**index, step = 0, 1**

**for char in s:**

**rows[index] += char**

**if index == 0:**

**step = 1**

**elif index == numRows - 1:**

**step = -1**

**index += step**

**return ''.join(rows)**

**s = "PAYPALISHIRING"**

**numRows = 3**

**print(convert(s, numRows))**

**Output: "PAHNAPLSIIGYIR"**

**7) def reverse\_integer(x):**

**if x < 0:**

**return int(str(x)[0] + str(x)[:0:-1])**

**else:**

**return int(str(x)[::-1]) if int(str(x)[::-1]) < 2\*\*31 - 1 else 0**

**x = 123**

**print(reverse\_integer(x))**

**Output: 321**

**8) import re**

**def myAtoi(s: str) -> int:**

**s = s.strip()**

**num = re.match('[\+\-]?\d+', s)**

**return max(min(int(num[0]) if num else 0, 2\*\*31 - 1), -2\*\*31)**

**9) def is\_palindrome(x):**

**return str(x) == str(x)[::-1]**

**x = 121**

**print(is\_palindrome(x))**