ASSIGNMENT-1 DAA

1. Two Sum:

```
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
    return []
print(two_sum([2,7,11,15], 9))  # Output: [0, 1]
print(two_sum([3,2,4], 6))      # Output: [1, 2]
print(two_sum([3,3], 6))       # Output: [0, 1]
```

2. Add Two Numbers:

```
 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, carry = dummy_head, 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


 List 1: 2 -> 4 -> 3
 List 2: 5 -> 6 -> 4
Result: 7 -> 0 -> 8
```

3. Longest Substring without Repeating Characters:

```
def length_of_longest_substring(s):
    char_set = set()
    left = 0
    max_length = 0

    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_length = max(max_length, right - left + 1)

    return max_length
print(length_of_longest_substring("abcabcbb"))  # Output: 3
print(length_of_longest_substring("bbbbb"))     # Output: 1
print(length_of_longest_substring("pwwkew"))    # Output: 3
```

4. Median of Two Sorted Arrays:

```
 def find_median_sorted_arrays(nums1, nums2):
    A, B = nums1, nums2
    if len(A) > len(B):
        A, B = B, A
    m, n = len(A), len(B)

    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 B[j-1] > A[i]:
            imin = i + 1
        elif i > 0 and A[i-1] > B[j]:
            imax = i - 1
        else:
            if i == 0: max_of_left = B[j-1]
            elif j == 0: max_of_left = A[i-1]
            else: max_of_left = max(A[i-1], B[j-1])

            if (m + n) % 2 == 1:
                return max_of_left

            if i == m: min_of_right = B[j]
            elif j == n: min_of_right = A[i]
            else: min_of_right = min(A[i], B[j])

            return (max_of_left + min_of_right) / 2.0
print(find_median_sorted_arrays([1, 3], [2]))         # Output: 2.0
print(find_median_sorted_arrays([1, 2], [3, 4]))      # Output: 2.5
```

5. Longest Palindromic Substring:

```
def longest_palindrome(s):
    if len(s) == 0:
        return ""
    start = 0
    end = 0
    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        max_len = max(len1, len2)
        if max_len > end - start:
            start = i - (max_len - 1) // 2
            end = i + max_len // 2
    return s[start:end + 1]

def expand_around_center(s, left, right):
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return right - left - 1
print(longest_palindrome("babad"))  # Output: "bab" or "aba"
print(longest_palindrome("cbbd"))   # Output: "bb"
```

6. Zigzag Conversion:

```
def convert(s, numRows):
    if numRows == 1:
        return s

    rows = [''] * min(numRows, len(s))
    cur_row = 0
    going_down = False

    for c in s:
        rows[cur_row] += c
        if cur_row == 0 or cur_row == numRows - 1:
            going_down = not going_down
        cur_row += 1 if going_down else -1

    return ''.join(rows)
print(convert("PAYPALISHIRING", 3))  # Output: "PAHNAPLSIIGYIR"
print(convert("PAYPALISHIRING", 4))  # Output: "PINALSIGYAHRPI"
print(convert("A", 1))              # Output: "A"
```

7. Reverse Integer:

```
 def reverse(x):
    sign = -1 if x < 0 else 1
    x *= sign
    reversed_x = 0

    while x:
        reversed_x = reversed_x * 10 + x % 10
        x //= 10

    reversed_x *= sign

    if reversed_x < -2**31 or reversed_x > 2**31 - 1:
        return 0
    return reversed_x
print(reverse(123))    #Output: 321
print(reverse(-123))  # Output: -321
print(reverse(120))   # Output: 21
```

8. String to Integer (atoi):

```
def myAtoi(s):
    s = s.lstrip()
    if not s:
        return 0

    sign = 1
    index = 0
    if s[0] in ['-', '+']:
        if s[0] == '-':
            sign = -1
        index += 1

    result = 0
    while index < len(s) and s[index].isdigit():
        result = result * 10 + int(s[index])
        index += 1

    result *= sign
    if result < -2**31:
        return -2**31
    if result > 2**31 - 1:
        return 2**31 - 1
    return result
print(myAtoi("42"))            # Output: 42
print(myAtoi("   -42"))        # Output: -42
print(myAtoi("4193 with words"))  # Output: 4193
```

9. Palindrome Number:

```
def is_palindrome(x):
    if x < 0:
        return False
    return str(x) == str(x)[::-1]
print(is_palindrome(121))  # Output: True
print(is_palindrome(-121)) # Output: False
print(is_palindrome(10))   # Output: False
```

10. Regular Expression Matching:

```
def is_match(s, p):
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True

    # Initialize dp[0][j] for patterns like a*, a*b*, a*b*c*
    for j in range(2, n + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if p[j - 1] == '*':
                # '*' Matches zero preceding element
                dp[i][j] = dp[i][j - 2]
                # '*' Matches one or more preceding element
                if p[j - 2] == s[i - 1] or p[j - 2] == '.':
                    dp[i][j] = dp[i][j] or dp[i - 1][j]
            else:
                if p[j - 1] == s[i - 1] or p[j - 1] == '.':
                    dp[i][j] = dp[i - 1][j - 1]

    return dp[m][n]
print(is_match("aa", "a"))     # Output: False
print(is_match("aa", "a*"))    # Output: True
print(is_match("ab", ".*"))    # Output: True
```