Set A Problem 1 (Easy)

In [ ]:

```
# Instructions:
# Before typing:
# 1. Open the command palette (Ctrl+shift+P on Windows, Command+Shift+P on Mac)
#    and search for: Enable Saving Files
# 2. Hit the RUN button to open the terminal, type '../../scripts/recordTerminal.sh'
# 3. Press Ctrl+alt+R on Windows(Ctrl+Option+R on Mac) to start recording your keystrokes
# Begin typing!

class Solution(object):
    """
    You are given a string `s` consisting of lowercase English letters.
    A duplicate removal consists of choosing two adjacent and equal letters and removing them.
    We repeatedly make duplicate removals on `s` until we no longer can.
    Return the final string after all such duplicate removals have been made.
    It can be proven that the answer is unique.
    """

    # Example 1:
    """
    Input: s = "abbaca"
    Output: "ca"
    Explanation:
    For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal,
    and this is the only possible move.  The result of this move is that the string is "aaca",
    of which only "aa" is possible, so the final string is "ca".
    """
    # Example 2:
    """
    Input: s = "azxxzy"
    Output: "ay"
    """

    # Constraints:
    """
    1 <= s.length <= 10^5
    `s` consists of lowercase English letters.
    """

    def removeDuplicates(self, s):
        """
        :type s: str
        :rtype: str
        """
        return ""



class Test(object):
    def test_removeDuplicates(self):

        # Test 1 begins
        print("======Test 1======\n")
        solution = Solution()
        answer1 = solution.removeDuplicates("abbaca")
        print("Input: \"abbaca\"\nExpected: \"ca\"\nOutput: \"", end = '')
        print(answer1, end = '\"\n\n')
        if answer1 == "ca":
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print('\n')

        # Test 2 begins
        print("=======Test 2=======\n")
        solution = Solution()
        answer2 = solution.removeDuplicates("azxxzy")
        print("Input: \"azxxzy\"\nExpected: \"ay\"\nOutput: \"", end = '')
        print(answer2, end = '\"\n\n')
        if answer2 == "ay":
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")

if __name__ == '__main__':
```

```
    test = Test()
    test.test_removeDuplicates()
```

Set A Problem 2 (Easy)

In [ ]:

```
class TreeNode(object):
        def __init__(self, val=0, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right
class Solution(object):
    """
    Given the root of a binary tree, return all root-to-leaf paths in given format.
    You can return the paths in any order.
    A leaf is a node with no children.
    """

    # Example 1:
    """
            [1]
          /     \
         /       \
       [2]       [3]
         \
          \
          [5]
    Input: root = [1,2,3,null,5]
    Output: ["1->2->5","1->3"]
    """
    # Example 2:
    """
    Input: root = [1]
    Output: ["1"]
    """

    #Constraints:
    """
    The number of nodes in the tree is in the range [1,100].
    -100 <= Node.val <= 100
    """

    #Definition for a binary tree node.
    """
    class TreeNode(object):
        def __init__(self, val=0, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right
    """

    def binaryTreePaths(self, root):
        """
        :type root: TreeNode
        :rtype: List[str]
        """
        return []



class Test(object):
    def test_binaryTreePaths(self):

        # Test 1 begins
        print("=======Test 1=======\n")
        solution = Solution()
        inputTree1 = TreeNode(1,TreeNode(2,None,TreeNode(5)),TreeNode(3))
        answer1=solution.binaryTreePaths(inputTree1)
        answer1.sort()
        print("Input: [1,2,3,null,5]\nExpected: [\'1->2->5\',\'1->3\']\nOutput: ", end = '')
        print(answer1, end = '\n\n')
        if answer1 == ["1->2->5","1->3"]:
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print('\n')

        # Test 2 begins
        print("=======Test 2=======\n")
```

```
        solution = Solution()
        inputTree2 = TreeNode(1)
        answer2 = solution.binaryTreePaths(inputTree2)
        answer2.sort()
        print("Input: [1]\nExpected: [\'1\']\nOutput: ", end = '')
        print(answer2,end = '\n\n')
        if answer2 == ["1"]:
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")


if __name__ == '__main__':
    test=Test()
    test.test_binaryTreePaths()
```

Set A Problem 3 (Medium)

In [ ]:

```
# Instructions:
# When you finish:
# 1. In the same terminal, type '../../scripts/saveTerminal.sh'
# 2. Open the command palette (Ctrl+shift+P on Windows, Command+Shift+P on Mac)
#    and search for: Disable Saving Files
# 3. Press Ctrl+alt+R on Windows(Ctrl+Option+R on Mac) to stop recording your keystrokes

class Solution(object):
    """
    Given an m x n matrix `board` where each cell is a battleship 'X' or empty '.',
    return the number of the battleships on `board`.
    Battleships can only be placed horizontally or vertically on `board`.
    In other words, they can only be made of the shape 1 x k (1 row, k columns)
    or k x 1 (k rows, 1 column), where k can be of any size.
    We ensure that at least one horizontal or vertical cell separates between two battleships
    (i.e., there are no adjacent battleships).
    """
    # Example 1:
    """
    -------------------------
    |     |     |     |     |
    |  X  |     |     |  X  |
    -------------------------
    |     |     |     |     |
    |     |     |     |  X  |
    -------------------------
    |     |     |     |     |
    |     |     |     |  X  |
    -------------------------

    Input: board = [["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]]
    Output: 2
    """
    # Example 2:
    """
    Input: board = [["."]]
    Output: 0
    """

    # Constraints:
    """
    - m == board.length
    - n == board[i].length
    - 1 <= m, n <= 200
    - board[i][j] is either '.' or 'X'
    """

    def countBattleships(self, board):
        """
        :type board: List[List[str]]
        :rtype: int
        """
        return 0


class Test(object):
    def test_countBattleShips(self):

        # Test 1 begins
        print("=======Test 1=======\n")
```

```
        solution = Solution()
        answer1 = solution.countBattleships([["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]])
        print("Input: [[\"X\",\".\",\".\",\"X\"], [\".\",\".\",\".\",\"X\"], [\".\",\".\",\".\",\"X\"]]\nExpected: 2\nOutput:
        if answer1 == 2:
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print('\n')

        # Test 2 begins
        print("=======Test 2=======\n")
        solution = Solution()
        answer2 = solution.countBattleships([["."]])
        print("Input: [[\".\"]]\nExpected: 0\nOutput: " + str(answer2) + '\n')
        if answer2 == 0:
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")

if __name__ == '__main__':
    test=Test()
    test.test_countBattleShips()
```

Set B Problem 1 (Easy)

In [ ]:

```
# Instructions:
# Before typing:
# 1. Open the command palette (Ctrl+shift+P on Windows, Command+Shift+P on Mac)
#    and search for: Enable Saving Files
# 2. Hit the RUN button to open the terminal, type '../../scripts/recordTerminal.sh'
# 3. Press Ctrl+alt+R on Windows(Ctrl+Option+R on Mac) to start recording your keystrokes
# Begin typing!

class Solution(object):
    """
    Given an array of distinct integers `arr`, find all pairs of
    elements with the minimum absolute difference of any two elements.
    Return a list of pairs in ascending order(with respect to pairs), each pair [a, b] follows
    - a, b are from `arr`
    - a < b
    - b - a equals to the minimum absolute difference of any two elements in `arr`
    """
    # Example 1:
    """
    Input: arr = [4,2,1,3]
    Output: [[1,2],[2,3],[3,4]]
    Explanation: The minimum absolute difference is 1.
    List all pairs with difference equal to 1 in ascending order.
    """
    # Example 2:
    """
    Input: arr = [1,3,6,10,15]
    Output: [[1,3]]
    """
    # Example 3:
    """
    Input: arr = [3,8,-10,23,19,-4,-14,27]
    Output: [[-14,-10],[19,23],[23,27]]
    """
    # Constraints:
    """
    - 2 <= arr.length <= 10^5
    - -10^6 <= arr[i] <= 10^6
    """
    def minimumAbsDifference(self, arr):
        """
        :type arr: List[int]
        :rtype: List[List[int]]
        """
        return []



class Test(object):
    def test_minimumAbsDifference(self):

        # Test 1 begins
```

```
        print("======Test1======\n")
        solution = Solution()
        answer1 = solution.minimumAbsDifference([4,2,1,3])
        answer1.sort()
        print("Input: [4, 2, 1, 3]\nExpected: [[1, 2], [2, 3], [3, 4]]\nOutput: ", end = '')
        print(answer1, end = '\n\n')
        if len(answer1) == 3 and answer1[0] == [1,2] and answer1[1] == [2,3] and answer1[2] == [3,4]:
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print("\n")

        # Test 2 begins
        print("======Test2======\n")
        solution = Solution()
        answer2 = solution.minimumAbsDifference([1,3,6,10,15])
        answer2.sort()
        print("Input: [1, 3, 6, 10, 15]\nExpected: [[1, 3]]\nOutput: ", end = '')
        print(answer2, end = '\n\n')
        if answer2 == [[1, 3]]:
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")
        print("\n")

        # Test 3 begins
        print("======Test3======\n")
        solution = Solution()
        answer3 = solution.minimumAbsDifference([3,8,-10,23,19,-4,-14,27])
        answer3.sort()
        print("Input: [3, 8, -10, 23, 19, -4, -14, 27]\nExpected: [[-14, -10], [19, 23], [23, 27]]\nOutput: ", end = '')
        print(answer3, end = '\n\n')
        if len(answer3) == 3  and answer3[0] == [-14,-10] and answer3[1] == [19,23] and answer3[2] == [23,27]:
            print("====Test 3 passed====")
        else:
            print("====Test 3 failed====")
        print("\n")


if __name__ == '__main__':
    test=Test()
    test.test_minimumAbsDifference()
```

Set B Problem 2 (Easy)

In [ ]:

```
class ListNode(object):
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution(object):
    """
    You are given the heads of two sorted linked lists *list1* and *list2*.
    Merge the two lists in one sorted list. Ths list should be made by splicing
    together the nodes of the first two lists.
    Return the head of the merged linked list.
    """

    # Example 1:
    """
            list1           [1]->[2]->[4]
            list2           (1)->(3)->(4)
            return   (1)->[1]->[2]->(3)->[4]->(4)

    Input: list1 = [1,2,4], list2 = [1,3,4]
    Output: [1,1,2,3,4,4]
    """
    # Example 2:
    """
    Input: list1 = [], list2 = []
    Output: []
    """
    # Example 3:
    """
    Input: list1 = [], list2 = [0]
    Output: [0]
    """
    #Constraints:
```

```
    """
    The number of nodes in both lists is in the range [0, 50].
    -100 <= Node.val <= 100
    Both list1 and list2 are sorted in non-decreasing order.
    """
    # Definition for singly-linked list.
    """
    class ListNode(object):
        def __init__(self, val=0, next=None):
            self.val = val
            self.next = next
    """

    def mergeTwoLists(self, list1, list2):
        """
        :type list1: Optional[ListNode]
        :type list2: Optional[ListNode]
        :rtype: Optional[ListNode]
        """
        return None



class Test(object):
    def test_mergeTwoLists(self):
        def print_list(head):
            # This function helps to convert a list object to a python list
            plist=[]
            while head:
                plist.append(head.val)
                head=head.next
            return plist



        # Test 1 begins
        print("=======Test 1=======\n")
        solution=Solution()
        inputList1 = ListNode(1, ListNode(2, ListNode(4)))
        inputList2 = ListNode(1, ListNode(3, ListNode(4)))
        answer1 = print_list(solution.mergeTwoLists(inputList1, inputList2))
        print("Input: List1 = [1, 2, 4] List2 = [1, 3, 4]\nExpected: [1, 1, 2, 3, 4, 4]\nOutput: ", end = '')
        print(answer1, end = '\n\n')
        if answer1 == [1, 1, 2, 3, 4, 4]:
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print('\n')

        # Test 2 begins
        print("=======Test 2=======\n")
        solution=Solution()
        inputList1 = None
        inputList2 = None
        answer2 = print_list(solution.mergeTwoLists(inputList1, inputList2))
        print("Input: List1 = [] List2 = []\nExpected: []\nOutput: ", end = '')
        print(answer2, end = '\n\n')
        if not answer2:
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")
        print('\n')

        # Test 3 begins
        print("=======Test 3=======\n")
        solution=Solution()
        inputList1 = None
        inputList2 = ListNode(0)
        answer3 = print_list(solution.mergeTwoLists(inputList1, inputList2))
        print("Input: List1 = [] List2 = [0]\nExpected: [0]\nOutput: ", end = '')
        print(answer3, end = '\n\n')
        if answer3 == [0]:
            print("====Test 3 passed====")
        else:
            print("====Test 3 failed====")
        print('\n')


if __name__ == '__main__':
```

```
    test=Test()
    test.test_mergeTwoLists()
```

Set B Problem 3 (Medium)

In [ ]:

```
# Instructions:
# When you finish:
# 1. In the same terminal, type '../../scripts/saveTerminal.sh'
# 2. Open the command palette (Ctrl+shift+P on Windows, Command+Shift+P on Mac)
#    and search for: Disable Saving Files
# 3. Press Ctrl+alt+R on Windows(Ctrl+Option+R on Mac) to stop recording your keystrokes

class Solution(object):
    """
    Given an m x n matrix, return all elements of the matrix in spiral order.
    """
    # Example 1:
    """
    -------------
    | 1 | 2 | 3 |
    -------------
    | 4 | 5 | 6 |
    -------------
    | 7 | 8 | 9 |
    -------------

    Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
    Output: [1,2,3,6,9,8,7,4,5]
    """
    # Example 2:
    """
    ------------------
    | 1 | 2 | 3 | 4 |
    ------------------
    | 5 | 6 | 7 | 8 |
    ------------------
    | 9 | 10| 11| 12|
    ------------------
    Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
    Output: [1,2,3,4,8,12,11,10,9,5,6,7]
    """
    # Constraints:
    """
    - m == matrix.length
    - n == matrix[i].length
    - 1 <= m, n <= 10
    - -100 <= matrix[i][j] <= 100
    """

    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        return []



class Test(object):
    def test_spiralOrder(self):

        # Test 1 begins
        print("=======Test 1=======\n")
        solution = Solution()
        answer1 = solution.spiralOrder([[1, 2, 3], [4, 5, 6],[7, 8, 9]])
        print("Input: matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]\nExpected: [1, 2, 3, 6, 9, 8, 7, 4, 5]\nOutput: ", end = '')
        print(answer1, end = "\n\n")
        if answer1 == [1, 2, 3, 6, 9, 8, 7, 4, 5]:
            print("====Test 1 passed====")
        else:
            print("====Test 1 failed====")
        print('\n')

        # Test 2 begins
        print("=======Test 2=======\n")
        solution = Solution()
        answer2 = solution.spiralOrder([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
        print("Input: matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]\nExpected: [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7
```

```
        print(answer2, end = "\n\n")
        if answer2 == [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6,7]:
            print("====Test 2 passed====")
        else:
            print("====Test 2 failed====")


if __name__ == '__main__':
    test=Test()
    test.test_spiralOrder()
```