P YOSHITHA
2211CS020607
AIML EPSILON

DAA HOLIDAY ASSIGNMENT

1.Remove duplicates from sorted list

```
# Definition for singly-linked list.
class ListNode(object):
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: Optional[ListNode]
        :rtype: Optional[ListNode]
        """
        current = head

        # Traverse the list
        while current and current.next:
            # If the current value equals the next value, skip the duplicate
            if current.val == current.next.val:
                current.next = current.next.next
            else:
                # Move to the next node if no duplicate
                current = current.next

        return head
```

2.Find a Peek Element

```
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        left, right = 0, len(nums) - 1

        while left < right:
            mid = (left + right) // 2

            # Compare mid with its next element
            if nums[mid] > nums[mid + 1]:
                # Peak is in the left half (including mid)
                right = mid
            else:
                # Peak is in the right half
                left = mid + 1

        # Left and right converge to the peak index
        return left
```

3.Binary Tree: Inorder Traversal

```
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[int]
        """
        def helper(node, result):
            if node:
                # Traverse left subtree
                helper(node.left, result)
                # Visit root
                result.append(node.val)
                # Traverse right subtree
                helper(node.right, result)

        result = []
        helper(root, result)
        return result
```

4.Valid Paranthesis

```
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        # Stack to keep track of opening brackets
        stack = []

        # Dictionary to match closing brackets with opening ones
        mapping = {")": "(", "}": "{", "]": "["}

        # Iterate through each character in the string
        for char in s:
            if char in mapping: # If it's a closing bracket
                # Pop the top of the stack, if stack is empty return a mismatch
                top_element = stack.pop() if stack else '#'

                # If the top element of the stack does not match the expected
opening bracket
                if mapping[char] != top_element:
                    return False
            else:
                # If it's an opening bracket, push it to the stack
                stack.append(char)
```

```
        # If the stack is empty, all brackets matched correctly, else there's an
unmatched opening bracket
        return not stack
```

5.Merge Two Sorted Lists

```
# Definition for singly-linked list.
class ListNode(object):
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution(object):
    def mergeTwoLists(self, list1, list2):
        """
        :type list1: Optional[ListNode]
        :type list2: Optional[ListNode]
        :rtype: Optional[ListNode]
        """
        # Create a dummy node to act as the head of the new list
        dummy = ListNode(-1)
        current = dummy

        # Traverse both lists and merge
        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

        # Attach any remaining nodes
        current.next = list1 if list1 else list2

        # Return the merged list starting from the next of dummy
        return dummy.next
```

6. Find the Index of the First occurrence in a String

```
class Solution(object):
    def strStr(self, haystack, needle):
        """
        :type haystack: str
        :type needle: str
        :rtype: int
        """
```

```
        # If the needle is an empty string, return 0
        if not needle:
            return 0

        # Iterate over the haystack to find the first occurrence of needle
        for i in range(len(haystack) - len(needle) + 1):
            if haystack[i:i + len(needle)] == needle:
                return i # Return the index of the first occurrence

        # If no occurrence is found, return -1
        return -1
```

7.N - Queens

```
class Solution(object):
    def solveNQueens(self, n):
        """
        :type n: int
        :rtype: List[List[str]]
        """
        def is_safe(board, row, col):
            # Check if placing the queen at (row, col) is safe
            for i in range(row):
                # Check column and diagonals
                if board[i] == col or \
                   board[i] - i == col - row or \
                   board[i] + i == col + row:
                    return False
            return True

        def solve(board, row):
            # If all queens are placed, add the board configuration to results
            if row == n:
                result.append(['.' * i + 'Q' + '.' * (n - i - 1) for i in board])
                return

            for col in range(n):
                if is_safe(board, row, col):
                    board[row] = col # Place queen
                    solve(board, row + 1) # Recur for the next row
                    board[row] = -1 # Backtrack

        result = []
        solve([-1] * n, 0) # Initialize the board and start solving from the
first row
        return result
```

8. Largest Number

```
from functools import cmp_to_key

class Solution(object):
    def largestNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: str
        """
        # Custom comparator
        def compare(x, y):
            if x + y > y + x:
                return -1
            elif x + y < y + x:
                return 1
            else:
                return 0

        # Convert numbers to strings for custom sorting
        nums = list(map(str, nums))
        # Sort using the custom comparator
        nums.sort(key=cmp_to_key(compare))
        # Concatenate sorted numbers
        result = ''.join(nums)
        # Handle the case where the result is all zeros
        return '0' if result[0] == '0' else result
```

9.Bitwise and of Number Range e

```
class Solution(object):
    def rangeBitwiseAnd(self, left, right):
        # Shift left and right until they are equal
        shift = 0
        while left < right:
            left >>= 1
            right >>= 1
            shift += 1

        # Restore the common prefix
        return left << shift
```

10.Square Root

```cpp
class Solution {
public:
    int mySqrt(int x) {
        if (x == 0) return 0; // Special case for 0

        int left = 1, right = x;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            long long square = (long long)mid * mid; // To avoid overflow

            if (square == x) {
                return mid;
            } else if (square < x) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        // After the loop, right will be the integer part of the square root
        return right;
    }
};
```

P. Yoshitha

2211CS020607

AIML - EPSILON