45. Jump Game II

```
class Solution:
    def jump(self, nums: List[int]) -> int:
        steps = 0
        current_end = 0
        farthest = 0

        for i in range(len(nums) - 1):
            farthest = max(farthest, i + nums[i])
            if i == current_end:
                steps += 1
                current_end = farthest

        return steps
```

55. Jump Game

```
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        max_reach = 0
        for i, num in enumerate(nums):
            if i > max_reach:
                return False
            max_reach = max(max_reach, i + num)
        return True
```

56. Merge Intervals

```python
class Solution:
    def merge(self, interval: List[List[int]]) -> List[List[int]]:
        interval.sort(key = lambda x:x[0])
        n = len(interval)
        i = 0
        ans = []
        while i <= n-1:
            start = interval[i][0]
            end = interval[i][1]
            while i < n-1 and end >= interval[i+1][0]:
                end = max(end,interval[i+1][1])
                start = min(start,interval[i+1][0])
                i+=1
            ans.append([start,end])
            i+=1
        return ans

# from typing import List

# class Solution:
#     def merge(self, intervals: List[List[int]]) -> List[List[int]]:
#         if not intervals:
#             return []

#         intervals.sort(key=lambda x: x[0])
#         merged = [intervals[0]]

#         for curr in intervals[1:]:
#             last = merged[-1]
#             if curr[0] <= last[1]:  # Overlap
#                 last[1] = max(last[1], curr[1])
#             else:
#                 merged.append(curr)

#         return merged
```

57. Insert Interval

```
from typing import List

class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]) -> List[Li
st[int]]:
        result = []
        i = 0
        n = len(intervals)

        # Step 1: Add all intervals before newInterval
        while i < n and intervals[i][1] < newInterval[0]:
            result.append(intervals[i])
            i += 1

        # Step 2: Merge all overlapping intervals with newInterval
        while i < n and intervals[i][0] <= newInterval[1]:
            newInterval[0] = min(newInterval[0], intervals[i][0])
            newInterval[1] = max(newInterval[1], intervals[i][1])
            i += 1
        result.append(newInterval)

        # Step 3: Add all remaining intervals
        while i < n:
            result.append(intervals[i])
            i += 1

        return result
```

135. Candy

```
class Solution:
    def candy(self, ratings: List[int]) -> int:
        n = len(ratings)
        candies = [1] * n

        # Left to right
        for i in range(1, n):
            if ratings[i] > ratings[i - 1]:
                candies[i] = candies[i - 1] + 1

        # Right to left
        for i in range(n - 2, -1, -1):
            if ratings[i] > ratings[i + 1]:
                candies[i] = max(candies[i], candies[i + 1] + 1)

        return sum(candies)
```

435. Non-overlapping Intervals

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        intervals.sort(key = lambda x:x[1])
        if not intervals:return []
        merged = [intervals[0]]
        remove = 0
        for cur in intervals[1:]:
            last = merged[-1]
            if cur[0] < last[-1]:
                print(cur)
                remove +=1
            else:
                merged.append(cur)
        return  remove
```

455. Assign Cookies

```
class Solution:
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        i = 0
        j = 0
        ans = 0
        g.sort()
        s.sort()
        while i<len(g) and j < len(s):
            if g[i] <= s[j]:
                ans+=1
                i+=1
            j+=1
        return ans
```

678. Valid Parenthesis String

```
class Solution:
    def checkValidString(self, s: str) -> bool:
        low = 0   # Min number of open brackets
        high = 0  # Max number of open brackets

        for ch in s:
            if ch == '(':
                low += 1
                high += 1
            elif ch == ')':
                low -= 1
                high -= 1
            else:  # '*'
                low -= 1   # could be ')'
                high += 1  # could be '('

            if high < 0:
                return False  # Too many closing ')'

            if low < 0:
                low = 0  # We can't have less than 0 open brackets

        return low == 0
```

860. Lemonade Change

```
class Solution:
    def lemonadeChange(self, bills: List[int]) -> bool:
        five, ten = 0, 0

        for bill in bills:
            if bill == 5:
                five += 1
            elif bill == 10:
                if five == 0:
                    return False
                five -= 1
                ten += 1
            else:  # bill == 20
                if ten > 0 and five > 0:
                    ten -= 1
                    five -= 1
                elif five >= 3:
                    five -= 3
                else:
                    return False
        return True
```