# CS-218-HW2-Basic

Lakhan Kumar Sunilkumar

10 April 2025

**Student ID: 862481700**

## 1 A. Seating Arrangements: Codeforce Submission ID: 316016144

*Explanation:*

So A person is only happy when they sit next to a family member in a pair or sit alone in a row. Now We go through all possible numbers of family pairs ($x$) that can be seated in $r$ rows. For each $x$, we check the leftover people and the remaining rows. Then, check how many singles can be seated alone rather than sitting with a different family member which indeed breaks the happiness condition. The final result is the maximum total number of happy people that can be covered among all possibilities and combinations.

*Code Structure*

```
Function MinimizeMovesToSort(n, arr):
    Initialize a map to track the length of consecutive sequences
    set longest_sequence_length = 0

    For each value v in arr:
        If (v - 1) exists in map:
            seq_length = length of sequence ending at (v - 1) + 1
        Else:
            seq_length = 1

        Update the map for v with seq_length
        Update longest_sequence_length to maximum of itself and seq_length

    minimum_moves_required = n - longest_sequence_length

    Print minimum_moves_required
```

*Time Complexity:*

For each test case, we loop up to $\mathcal{O}(r)$ (maximum number of rows), where $r \leq 500$. Preprocessing steps like summing array elements take $\mathcal{O}(n)$, where $n \leq 100$. Thus, the complexity per test case is $\mathcal{O}(r)$. Overall, the time complexity for $t$ test cases is $\mathcal{O}(t \times r)$.

*Reference:*

https://www.geeksforgeeks.org/puzzle-sitting-arrangement/

## 2 B. Artistic Swimming: Codeforce Submission ID: 316013960

*Explanation:*

The plan is to form the max number of swimming trios groups consisting of 1 male and 2 females, and the condition is that the male is always taller than both selected females. First, we should sort both male and female height arrays. Then, using a greedy two-pointer approach, we iterate over each male and try to find the smallest pair of vacant females who are both shorter than the male that is selected. If found, we make and increment the count of the trio and move the female pointer forward by 2 as 2 females are selected. This shows that pairing is done for shortest compatible females first.

*Code Structure*

```
Sort the maleHeights array in increasing order.
Sort the femaleHeights array in increasing order.

While malePointer < size of maleHeights AND femalePointer + 1 < size of femaleHeights:
    If maleHeights[malePointer] > femaleHeights[femalePointer]
       AND maleHeights[malePointer] > femaleHeights[femalePointer + 1]:
        i. Form a trio with the current male and the two females.
        ii. Increment trioCount by 1.
        iii. Move femalePointer forward by 2 (both females used).
    Else:
        i. Move femalePointer forward by 1 (current female not usable, try next).
    Move malePointer forward by 1 (use the next male).
Return trioCount.
```

*Time Complexity:*

- Sorting females array: $O(n \log n)$
- Sorting males array: $O(m \log m)$
- Greedy two-pointer scan: $O(n + m)$

*Overall Time Complexity:* $O(n \log n + m \log m)$

*References*

- https://www.geeksforgeeks.org/when-should-i-use-two-pointer-approach/
- https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutor

# 3 C. Number Candles: Codeforce Submission ID: 316013231

*Explanation:*

By using Greedy Algorithm:

The plan is to have a maximum number of candle digits by buying as much as the cheapest candles as possible. For each digit position (from left to right), replace it with the largest digit that still allows enough money to fill the remaining positions with the cheapest digit.

*Code Structure*

```
Find the minimum price among all digits: min_price = min(prices).
Calculate the maximum possible number of digits: length = monies // min_price.
For each position from 0 to length - 1:
    For digit d from 9 down to 1:
        i. Get cost_d = prices[d - 1].
        ii. Calculate remaining positions: rem_pos = length - pos - 1.
        iii. If rem_money - cost_d >= rem_pos * min_price:
             a. Append digit d to result.
             b. Deduct cost_d from rem_money.
             c. Break the inner loop (move to next position).
After all positions are filled, print it.
```

*Time Complexity:*

O(n), where n: maximum number of digits ($n = \lfloor \frac{M}{\text{min\_price}} \rfloor$).

Since the outer loop (for each digit) and the nested loop (constant 9 tries) are independent, the total complexity remains **O(n)**.

*References:*

Prof Yan Gu's slides: 05-Greedy i.e Homework Deadlines and Buying Gifts example