## October 7

### Easy Greedy Algorithms Problems

# Problem 1: The Lemonade Change Problem

**Problem Statement:**

At a lemonade stand, each lemonade costs $5. Customers are standing in a queue to buy from you and order one at a time (in the order specified by bills). Each customer will only buy one lemonade and pay with either a $5, $10, or $20 bill. You must provide the correct change to each customer so that the net transaction is that the customer pays $5.

Note that you do not have any change in hand at first.

Given an integer array bills where bills[i] is the bill the ith customer pays, return true if you can provide every customer with the correct change, or false otherwise.

**Link to problem:**

https://leetcode.com/problems/lemonade-change/description/

---

**Example 1:**

- **Input:** `bills = [5, 5, 5, 10, 20]`
- **Output:** `true`
  **Explanation:**
  From the first 3 customers, we collect three $5 bills in order.
  From the fourth customer, we collect a $10 bill and give back a $5.
  From the fifth customer, we give a $10 bill and a $5 bill. Since all customers got correct change, we output true.

**Example 2:**

- **Input:** `bills = [5, 5, 10, 10, 20]`
- **Output:** `false`
  **Explanation:**
  From the first two customers in order, we collect two $5 bills.
  For the next two customers in order, we collect a $10 bill and give back a $5 bill.
  For the last customer, we cannot give the change of $15 back because we only have two $10 bills. Since not every customer received the correct change, the answer is false.

---

**Solution:**

```
class Solution {
   public boolean lemonadeChange(int[] bills) {
      int five = 0, ten = 0; // Initialize counts for $5 and $10 bills
      for (int i = 0; i < bills.length; i++) {
         if (bills[i] == 5) {
```

```
            five++; // Increment count for $5 bills
        } else if (bills[i] == 10) {
            if (five > 0) {
                ten++; // Increment count for $10 bills
                five--; // Give $5 change
            } else {
                return false; // Cannot provide change
            }
        } else { // bills[i] == 20
            if (five > 0 && ten > 0) {
                ten--; // Give one $10 and one $5 as change
                five--;
            } else if (five >= 3) {
                five -= 3; // Give three $5 bills as change
            } else {
                return false; // Cannot provide change
            }
        }
    }
    return true; // All customers received correct change
  }
}
```

---

**Explanation:**

- We maintain counters for the number of $5 and $10 bills.
- For each bill received:
  - If it's a $5 bill, we simply increment the count of $5 bills.
  - If it's a $10 bill, we check if we have a $5 bill to give as change. If yes, we decrement the count of $5 bills and increment the count of $10 bills.
  - If it's a $20 bill, we first try to give one $10 bill and one $5 bill as change. If that's not possible, we try to give three $5 bills instead. If we can't provide the correct change in either case, we return false.
- If we successfully process all customers, we return true.

---

**Time Complexity:**

- O(n), where n is the number of customers. We iterate through the array once.

**Space Complexity:**

- O(1), since we only use a constant amount of extra space for the counters.

---

# Problem 2: Assign Cookies

**Problem Statement:**

Assume you have g children with varying greed factors and s cookies with varying sizes. Your goal is to find the maximum number of children who can be satisfied with the given cookies. A child is satisfied if the size of the cookie assigned to them is greater than or equal to their greed factor.

Given two integer arrays g and s where g[i] is the greed factor of the ith child and s[j] is the size of the jth cookie, return the maximum number of children who can be satisfied.

**Link to problem:**

https://leetcode.com/problems/assign-cookies/description/

---

**Example 1:**

- **Input:** `g = [1, 2, 3], s = [1, 1]`
- **Output:** `1`
  **Explanation:** Only child 1 can be satisfied with a cookie of size 1.

**Example 2:**

- **Input:** `g = [1, 2], s = [1, 2, 3]`
- **Output:** `2`
  **Explanation:** Both children can be satisfied with cookies of size 1 and 2.

**Example 3:**

- **Input:** `g = [1, 2, 3], s = [3]`
- **Output:** `1`
  **Explanation:** Only child 3 can be satisfied with a cookie of size 3.

---

**Solution:**

```
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        Arrays.sort(g); // Sort the greed factors
        Arrays.sort(s); // Sort the cookie sizes
        int child = 0; // Pointer for children
        int cookie = 0; // Pointer for cookies

        // Loop until we exhaust either children or cookies
        while (child < g.length && cookie < s.length) {
            if (g[child] <= s[cookie]) {
                child++; // Child is satisfied, move to the next child
```

```
            }
            cookie++; // Move to the next cookie
        }

        return child; // Return the number of satisfied children
    }
}
```

---

**Explanation:**

- First, we sort both the greed factors of children and the sizes of cookies.
- We then use two pointers: one for children (child) and one for cookies (cookie).
- We loop through both arrays and check if the current cookie can satisfy the current child's greed factor:
    - If the current cookie satisfies the child, we increment both pointers to check the next child and the next cookie.
    - If the current cookie cannot satisfy the child, we only increment the cookie pointer to try the next cookie.
- The loop continues until we exhaust either the list of children or the list of cookies.
- Finally, we return the count of satisfied children.

---

**Time Complexity:**

- O(n log n + m log m), where n is the number of children and m is the number of cookies, due to the sorting operation.

**Space Complexity:**

- O(1), since we are using constant space for the pointers.

---