Day 9 / 100 :

Topic - Array, string

1 Problem statement: **Buddy String** (easy)

Given two strings s and goal, return true if you can swap two letters in s so the result is equal to goal, otherwise, return false.

Swapping letters is defined as taking two indices i and j (0-indexed) such that i != j and swapping the characters at s[i] and s[j].

For example, swapping at indices 0 and 2 in "abcd" results in "cbad".

Example 1:

Input: s = "ab", goal = "ba"
Output: true
Explanation: You can swap s[0] = 'a' and s[1] = 'b' to get "ba", which is equal to goal.
Example 2:

Input: s = "ab", goal = "ab"
Output: false
Explanation: The only letters you can swap are s[0] = 'a' and s[1] = 'b', which results in "ba" != goal.

Solutions :
Approach 1

Intuition:
The Intuition is to check if it is possible to swap two characters in string s to make it equal to string goal. It first handles the case where s and goal are identical by checking for duplicate characters. If they are not identical, it looks for the first pair of mismatched characters and tries swapping them to achieve equality. The code provides a solution by considering these two scenarios and returns true if swapping is successful, otherwise false.

Explanation:
1. First, it checks if s is equal to goal using the == operator. If they are equal, it means the strings are identical.

2. If s is equal to goal, the code creates a temporary set called temp to store the unique characters present in s. It does this by converting the string s to a set of characters using the set constructor.
3. The code then returns the result of the comparison temp.size() < goal.size(). This comparison checks if the size of the set temp (number of unique characters in s) is less than the size of the string goal. If it is, it means there are duplicate characters in s, and swapping any two of them would result in s becoming equal to goal. In this case, the function returns true; otherwise, it returns false.
4. If s is not equal to goal, the code proceeds to find the indices i and j such that s[i] and goal[i] are the first pair of characters that are different from each other when scanning from the left, and s[j] and goal[j] are the first pair of characters that are different from each other when scanning from the right.
5. The code uses a while loop to increment the i index from left to right until it finds a mismatch between s[i] and goal[i]. Similarly, it uses another while loop to decrement the j index from right to left until it finds a mismatch between s[j] and goal[j].
6. After finding the mismatched indices, the code checks if i is less than j. If it is, it means there is a pair of characters that can be swapped to make s equal to goal. In this case, the code uses the swap function to swap the characters s[i] and s[j].
7. Finally, the code checks if s is equal to goal after the potential swap. If they are equal, it means we have successfully swapped two characters to make s equal to goal, and the function returns true. Otherwise, it returns false.

```
class Solution {
public:
    bool buddyStrings(string s, string goal) {
        int n = s.length();
        if(s == goal){
            set<char> temp(s.begin(), s.end());
            return temp.size() < goal.size(); // Swapping same
characters
        }

        int i = 0;
        int j = n - 1;

        while(i < j && s[i] == goal[i]){
            i++;
```

```
        }

        while(j >= 0 && s[j] == goal[j]){
            j--;
        }

        if(i < j){
            swap(s[i], s[j]);
        }

        return s == goal;
    }
};
```

2 Problem statement: Pancake Sorting (Medium)

Given an array of integers arr, sort the array by performing a series of pancake flips.

In one pancake flip we do the following steps:

Choose an integer k where 1 <= k <= arr.length.
Reverse the sub-array arr[0...k-1] (0-indexed).
For example, if arr = [3,2,1,4] and we performed a pancake flip choosing k = 3, we reverse the sub-array [3,2,1], so arr = [1,2,3,4] after the pancake flip at k = 3.

Return an array of the k-values corresponding to a sequence of pancake flips that sort arr. Any valid answer that sorts the array within 10 * arr.length flips will be judged as correct.

Example 1:

Input: arr = [3,2,4,1]
Output: [4,2,4,3]
Explanation:
We perform 4 pancake flips, with k values 4, 2, 4, and 3.
Starting state: arr = [3, 2, 4, 1]
After 1st flip (k = 4): arr = [1, 4, 2, 3]
After 2nd flip (k = 2): arr = [4, 1, 2, 3]

After 3rd flip (k = 4): arr = [3, 2, 1, 4]
After 4th flip (k = 3): arr = [1, 2, 3, 4], which is sorted.
Example 2:

Input: arr = [1,2,3]
Output: []
Explanation: The input is already sorted, so there is no need to flip anything.
Note that other answers, such as [3, 3], would also be accepted.

Solutions :

Approach 1

Intuition
Why not just move the maximum element to the last??
Approach

- 1st flip to bring it to the front.
- 2nd flip to flip the whole array so that the maximum element moves to the last!
- And then pop out the last element i.e that maximum, and continue this process until your array becomes empty.

Complexity
Time complexity: O(n^2)
Space complexity: O(n)

```
class Solution {

    void flipArr(vector < int >& arr, int idx) {

        int i = 0, j = idx - 1;

        while ( i < j ) swap(arr[i++], arr[j--]);

        return;

    }

public:
    vector<int> pancakeSort(vector<int>& arr) {

        vector < int > ans;
```

```
        int n = arr.size();

        while ( arr.size() ) {

            n = arr.size();
            int maxIdx = max_element(arr.begin(), arr.end()) - 
arr.begin();

            flipArr(arr, maxIdx + 1);
            // for ( auto i : arr ) cout << i << " ";
            // cout << "\n";
            ans.push_back(maxIdx + 1);

            flipArr(arr, n);

            ans.push_back(n);

            arr.pop_back();

        }

        return ans;

    }
};
```