📘 Explanatory Notes – LeetCode 1402 (Reducing Dishes)

🔷 Problem Recap:

- Given an array `satisfaction[]` where each element = satisfaction level of a dish.
- We can choose **any subset** of dishes and cook them in **any order**.
- Each dish takes **1 unit of time**.
- Formula:

  ```
  Like-Time Coefficient = Σ (time[i] * satisfaction[i])
  ```

- Goal: Maximize this coefficient.
- If all satisfaction levels are negative → best is to cook nothing → result = 0.

🔷 Approach in Code:

Your code works with **sorting + suffix sum** to decide which dishes should be included.

🔷 Step-by-Step Explanation:

```
sort(satisfaction.begin(), satisfaction.end());
```

- Sort the array in **ascending order**.
- Why? → So that we can easily check whether including negative dishes helps or not.

```
vector<int> suff(n,0);
suff[n-1] = satisfaction[n-1];
```

- Create a **suffix sum array**.
- `suff[i]` = sum of all elements from index `i` to end.
- Example: if `satisfaction = [-9, -8, -1, 0, 5]`

  then `suff = [-13, -4, 4, 5, 5]`.

```
for(int i=n-2; i>=0; i--){
   suff[i] = satisfaction[i] + suff[i+1];
}
```

- Fill suffix sums.
- This helps check: *if we start cooking from this dish onwards, will the total contribution be non-negative?*

```
int idx = -1;
for(int i=0; i<n; i++){
   if(suff[i]>=0){
      idx = i;
      break;
   }
}
```

- Find the **first index (idx)** such that from this dish onwards, total satisfaction is non-negative.
- Why?
  - If `suff[i] < 0`, it means adding that dish (and everything after) decreases overall benefit.
  - If `suff[i] >= 0`, then it's safe to include dishes from here till end.

```
if(idx == -1) return 0;
```

- Edge case: If **no suffix is positive**, best answer = 0 (don't cook anything).

```
int max_sum=0;
int x=1;
for(int i=idx; i<n; i++){
    max_sum += satisfaction[i]*x;
    x++;
}
```

- Start from `idx` and calculate the **like-time coefficient**.
- `x` keeps track of time units (1, 2, 3, ...).
- Multiply each satisfaction value with its cooking time and accumulate.

🔷 Example Walkthrough

`satisfaction = [-1, -8, 0, 5, -9]`

1. Sort → `[-9, -8, -1, 0, 5]`
2. Suffix sums → `[-13, -4, 4, 5, 5]`
3. First index with non-negative suffix = `idx = 2` (`suff[2] = 4`).
4. Consider dishes from index 2 → `[ -1, 0, 5 ]`.
5. Compute coefficient:
   - `(-1 * 1) + (0 * 2) + (5 * 3) = -1 + 0 + 15 = 14`.

     ✅ Final Answer = `14`.

🔷 Key Idea Behind Code

- **Negative dishes can sometimes help** if followed by large positive dishes (e.g., -1 + 5).
- But if including a negative dish drags down the total (suffix < 0), discard it.

- Hence, find the **first safe index using suffix sums**, then calculate result.

🔷 Time & Space Complexity

- Sorting → `O(n log n)`
- Building suffix array → `O(n)`
- Final computation → `O(n)`
- **Overall = O(n log n)**
- Space = `O(n)` (for suffix array).

✅ **This is a clean and efficient solution using suffix sums to decide which prefix to discard.**
