# Project Idea: Reducing Dishes Problem

Team Members / Section 5

Abdallah ahmed mohamed abdallah 1000287901

abdelrhman mohamed hassan ibrahim 1000318228

Amr ahmed elsayed mohamed 1000287930

Abdelrhman Mohammed Ibrahim Aboalata 1000288306

Abdelrhman mohsen mohamed mohamed 1000287826

---

## 1. Problem Identification

### 1.1) Problem Description

Given a list of dishes, each dish has a satisfaction value that can be negative, zero, or positive.
Each dish takes one unit of time to cook, and its contribution is:

**satisfaction × cooking time**

The goal is to choose and order some dishes to get the **maximum total satisfaction**.
It is allowed to skip dishes that reduce the total result.

### 1.2) Input–Output Examples

**Input:** `[-1, -8, 0, 5, -9]`
**Output:** `14`

**Input:** `[4, 3, 2]`
**Output:** `20`

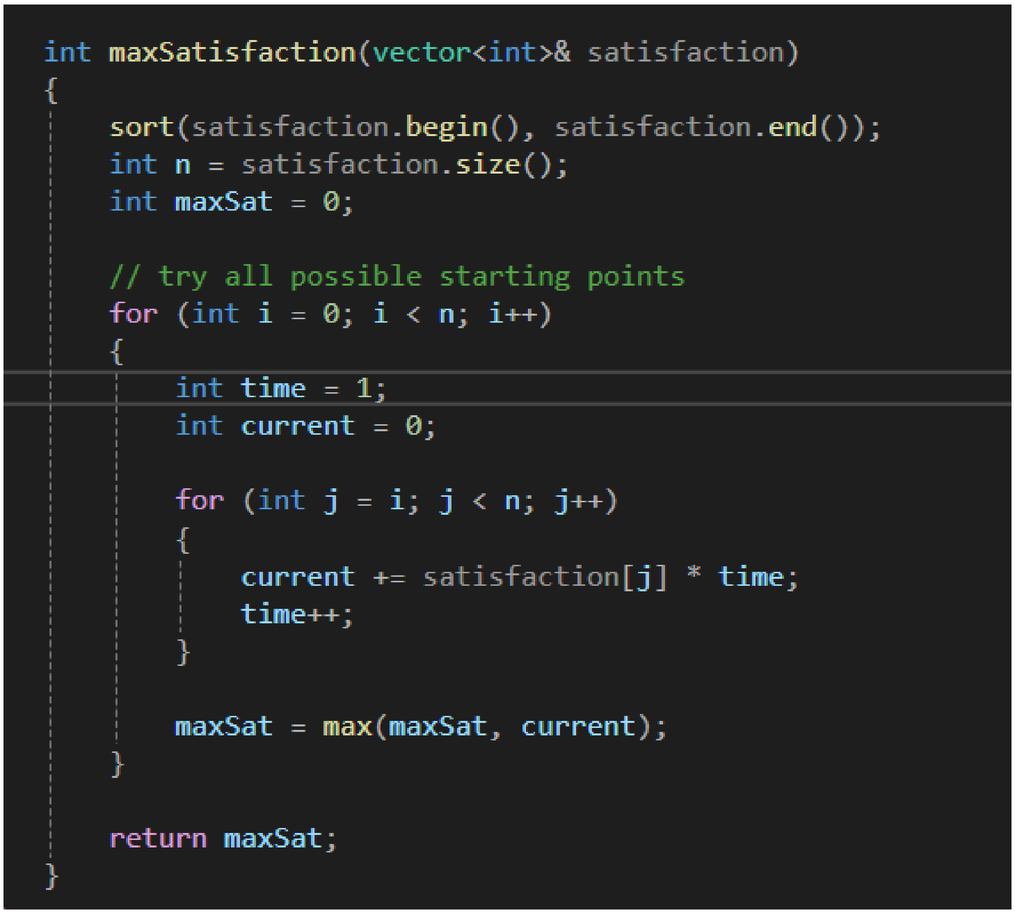
**Input:** `[-5, -3, -1]`
**Output:** `0`

# 2. Algorithm Development

## 2.1) Naive Algorithm

### Description

The naive approach sorts the array first, then tries all possible cooking orders by choosing different starting positions.
 For each case, it calculates the total satisfaction and keeps the maximum result.



```
int maxSatisfaction(vector<int>& satisfaction)
{
    sort(satisfaction.begin(), satisfaction.end());
    int n = satisfaction.size();
    int maxSat = 0;

    // try all possible starting points
    for (int i = 0; i < n; i++)
    {
        int time = 1;
        int current = 0;

        for (int j = i; j < n; j++)
        {
            current += satisfaction[j] * time;
            time++;
        }

        maxSat = max(maxSat, current);
    }

    return maxSat;
}
```

### Analysis

**Time Complexity:** O($n^2$)
 **Space Complexity:** O(1)

## Order analysis

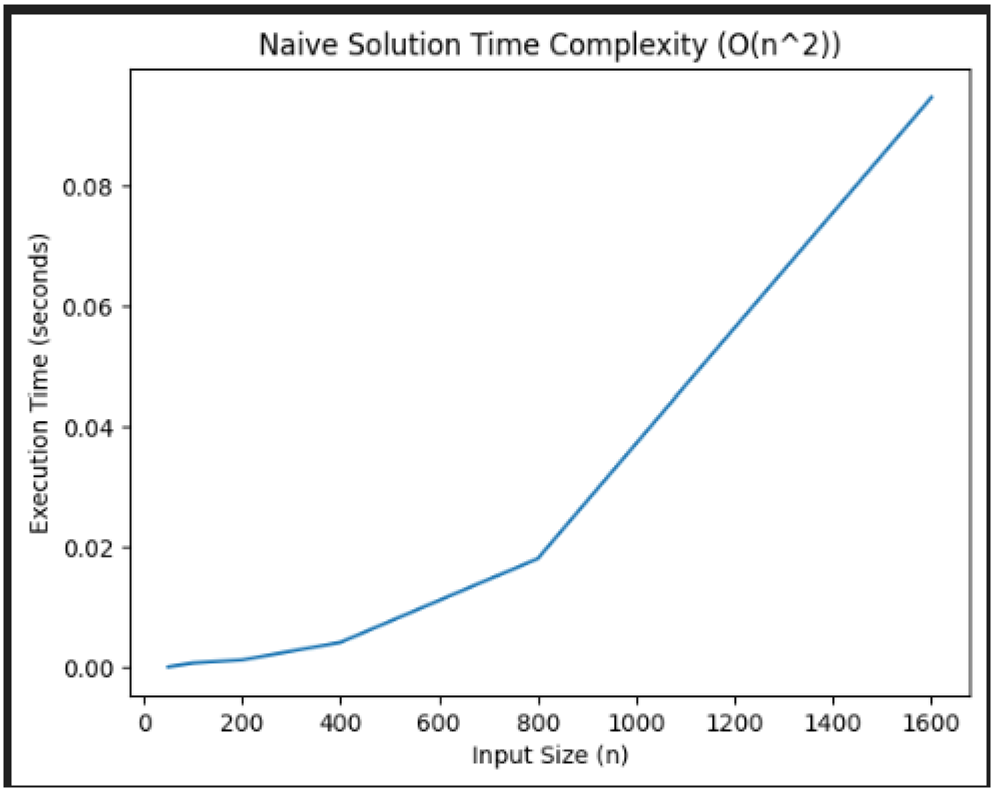
T(n) = O(n log n) + O(n^2) + O(1) + O(1) + O(1)

T(n) = O(n log n + n^2 + 3)

T(n) = O(n^2)

## Empirical Results – Naive

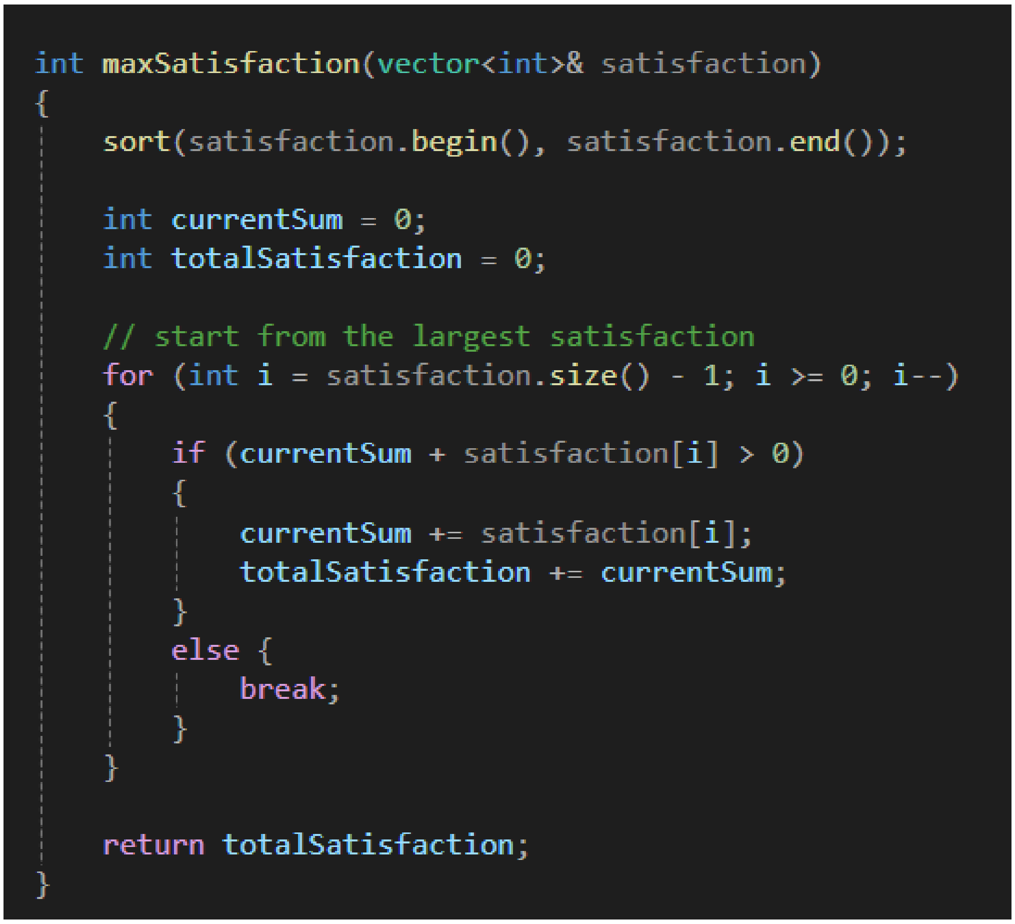
- Small input: ~0.08 ms
- Medium input: ~1 ms
- Large input: ~4 ms



## 2.2) Optimized Algorithm (Greedy)

### Description

After sorting, the algorithm starts from the largest satisfaction value and adds dishes one by one.
A dish is added only if it increases the total satisfaction.
The algorithm stops once adding a dish reduces the result.



```
int maxSatisfaction(vector<int>& satisfaction)
{
    sort(satisfaction.begin(), satisfaction.end());

    int currentSum = 0;
    int totalSatisfaction = 0;

    // start from the largest satisfaction
    for (int i = satisfaction.size() - 1; i >= 0; i--)
    {
        if (currentSum + satisfaction[i] > 0)
        {
            currentSum += satisfaction[i];
            totalSatisfaction += currentSum;
        }
        else {
            break;
        }
    }

    return totalSatisfaction;
}
```

### Analysis

**Time Complexity:** O(n log n)
**Space Complexity:** O(1)

### Order analysis

T(n) = O(n log n) + O(n) + O(1) + O(1) + O(1) + O(1)

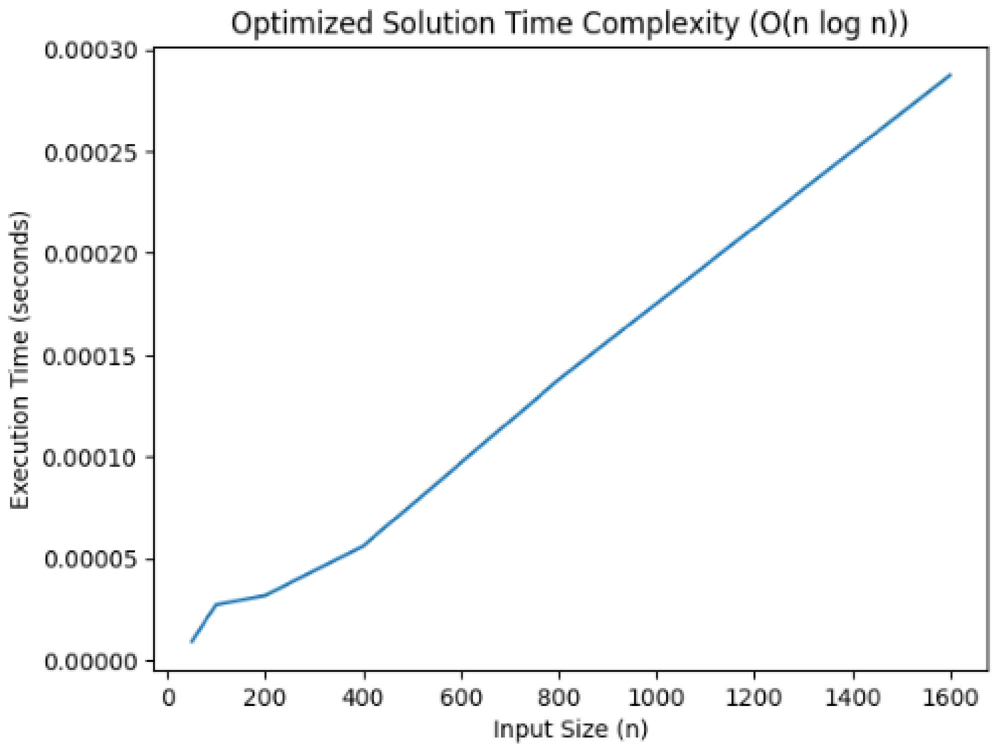
T(n) = O(n log n + n + 4)

T(n) = O(n log n)

## Empirical Results – Optimized

- Small input: ~0.01 ms
- Medium input: ~0.06 ms
- Large input: ~0.28 ms

---



## 3. Comparison Table

| Feature | Naive | Optimized |
|---|---|---|
| Time Complexity | $O(n^2)$ | $O(n \log n)$ |
| Space Complexity | $O(1)$ | $O(1)$ |
| Efficiency | Low | High |
| Redundancy | High | None |