perplexity

Halloumi Boxes - Revision Notes

Problem Statement

Given `n` boxes with numbers, determine if you can sort them in **non-decreasing order** using **reverse operations** on subarrays of **at most length K**.

Operation: Reverse any subarray of length ≤ K (can be done unlimited times)
Goal: Check if array can be sorted → Print "YES" or "NO"

Key Intuition

Case-Based Thinking

The problem breaks down into **two main cases**:

1. **Array is already sorted** → Answer is always **YES**
2. **Array is unsorted** → Depends on value of K

Core Insight: The Power of K

When K = 1

- Can only reverse subarrays of length 1 (single elements)
- Reversing a single element **doesn't change its position**
- **No power to move elements** to different positions
- If array is unsorted and K = 1 → Answer is **NO**

When K ≥ 2

- Can reverse subarrays of length 2 (which **swaps adjacent elements**)
- **KEY REALIZATION**: With length-2 reversals, you can move **any element to any position**
- Example: Move element from position i to position j by doing multiple adjacent swaps
- If K ≥ 2 → Answer is always **YES** (regardless of initial array state)

Algorithm Logic

```
if (array is already sorted):
    return "YES"
else if (K ≥ 2):
    return "YES"  // Can always sort with swaps
else:  // K = 1 and array unsorted
    return "NO"   // Cannot move elements
```

Implementation Approach

1. **Check if sorted**: Create a sorted copy and compare with original
2. **Apply logic**: Use the case-based reasoning above

```
vector<int> original = a;
sort(a.begin(), a.end());

if (original == a || k >= 2) {
    cout << "YES\n";
} else {
    cout << "NO\n";
}
```

Time Complexity

- **O(n log n)** for sorting to check if array is already sorted
- **O(n)** for comparison
- **Overall: O(n log n)**

Problem Pattern Recognition

Type: Constructive/Implementation with case analysis
Key Skill: Recognizing that K=2 gives enough power to sort any array
Similar Problems: Array manipulation with limited operations

Mental Framework for Similar Problems

1. **Identify the operation** and its limitations
2. **Find the minimum power needed** to solve the problem completely
3. **Check edge cases** where the power is insufficient
4. **Use case-based analysis** for clean solution logic

The core insight is understanding that **adjacent swaps (K≥2) are sufficient to sort any array**, while **K=1 provides no rearrangement power**.

⁂

[1] https://www.youtube.com/watch?v=3T2d0hjzdwA