Romerico David

10/30/2024

Quiz 3

(i) Exact Solution: Generate all possible ways to partition the array into two subsets of equal size and calculate the difference in their sums. This uses a brute-force approach to iterate through all possible subset combinations of size N/2 to determine the one with the minimal difference.

1. Generate all subsets of size N/2
2. For each subset:
    - Calculate the sum of the subset
    - Calculate the sum of the complementary subset
    - Update the min difference if the current difference is smaller

Pseudocode:

```
ExactMinDifferencePartition(int[] A):
        totalSum = sum(A)
        halfSize = N / 2
        minDiff  = Infinity
        bestSubsetA1, bestSubsetA2 = [], []

        allCombinations = generateCombinations(A, halfSize)

        For each subsetA1 in allCombinations:
                subsetA2 = Elements not in subsetA1
                sumA1 = sum(elements in subsetA1)
                sumA2 = totalSum - sumA1
                difference = abs(sumA1 - sumA2)
                If difference < minDifference:
```

minDifference = difference

bestSubsetA1 = subsetA1

bestSubsetA2 = subsetA2

Time Complexity: $O(2^n)$ for generating all subset combinations of size N/2, which is exponential. Then each subset requires calculating their own complement and sum which is O(1). The final time complexity is $O(2^n)$

Space Complexity: $O(2^n)$ to store all subset combinations and their sums

Do your answers correctly find the best partition, i.e., with the smallest difference?: The exact solution uses a brute-force approach which guareentes all possible subset combinations of size N/2 are explored. This guarantees finding the partition with the smallest possible difference in the sums between the two subsets.

(ii) Greedy Approach: Use greedy method to approximate the solution by trying to balance the sums while ensuring both subsets have exactly N/2 elements. Sort the elements in descending order then the algorithm prioritizes distributing the largest values first, which works because these values have the biggest impact on the overall balance between the two subsets. Each decision to add an element to the subset with the smaller sum is an attempt to keep the two subsets as close in total sum as possible. This lets us address the largest differences first (by placing larger elements in the subset needing more balance) and keep the total difference minimized.

Approach:

- Sort the array in decreasing order
- Initialize two empty subsets A1 and A2
- Iterate over the sorted array and assign each element to the subset with the current smaller sum and ensure neither subset exceeds N/2 elements

Pseudocode:

```
GreedyMinDifferencePartition(int[] A):

    Sort A in descending order

    Initialize empty subsets A1, A2

    Initialize sumA1 = 0, sumA2 = 0


    For each element in sorted A:

        If sumA1 <= sumA2 and size(A1) < N/2:

            Add element to A1

            sumA1 += element

        Else:

            Add element to A2

            sumA2 += element
```

Time Complexity: O(nlogn) due to the sorting step and the subset assignments require O(n) time

Space Complexity: O(n) to store the two subsets

Do your answers correctly find the best partition, i.e., with the smallest difference?: The greedy approach does not guarantee the smallest

possible difference but will provide a close approximation because it balances the large elements between the two subsets.

Example: Given the list {10, 9, 7, 6, 3, 1}, it would partition the lists into

A1 = { 10, 6, 3 } = 19

A2 = { 9, 7, 1} = 17

and the min difference is 2.

When the best solution is

A1 = { 10, 7, 1 } = 18

A2 = { 9, 6, 3} = 18

and the min difference is 2.