Project 3

---

1. Problem Statement

Pseudo-polynomial Partition

Given a set consisting of *n* integers [a_1, a_2, … a_n], you want to partition into two parts so that the sum of the two parts is equal. Suppose *s* = a_1 + a_2 … + a_n. The time complexity of your algorithm should be *O(ns)* or better.

The idea is to use dynamic programming (DP) to solve the partition problem. We begin by calculating the total sum **s** of the integers in the set; if **s** is odd, an equal partition is impossible. If **s** is even, we set the target sum to **s/2** and seek to determine if there exists a subset of elements that sums to this target. The DP table **dp[i][j]** indicates whether a subset sum **j** can be achieved using the first **i** elements of the array, where we update the table based on including or excluding each element. This approach runs in **O(ns)** time, where **s** is the total sum, making it a pseudo-polynomial solution suitable for moderate values of **n** and **s**.

2. Pseudocode

1. Calculate the sum of all elements in arr, let this be total Sum.
2. If total Sum is odd:
    - Return False.
3. Set target to total Sum / 2.
4. Initialize a 2D Boolean array dp of size (n + 1) x (target + 1).
    - dp[i][j] will be True if a subset with sum j can be achieved using the first i elements, otherwise False.
5. Set dp[i][0] = True for all i from 0 to n (a subset sum of 0 is always achievable by selecting no elements).
6. For each element i from 1 to n:
    - For each sum j from 1 to target:
    - If arr[i-1] <= j:
    - Set dp[i][j] = dp[i-1][j] OR dp[i-1][j - arr[i-1]] (include or exclude the element).
    - Else:
    - Set dp[i][j] = dp[i-1][j] (exclude the element).
7. Return dp[n][target].

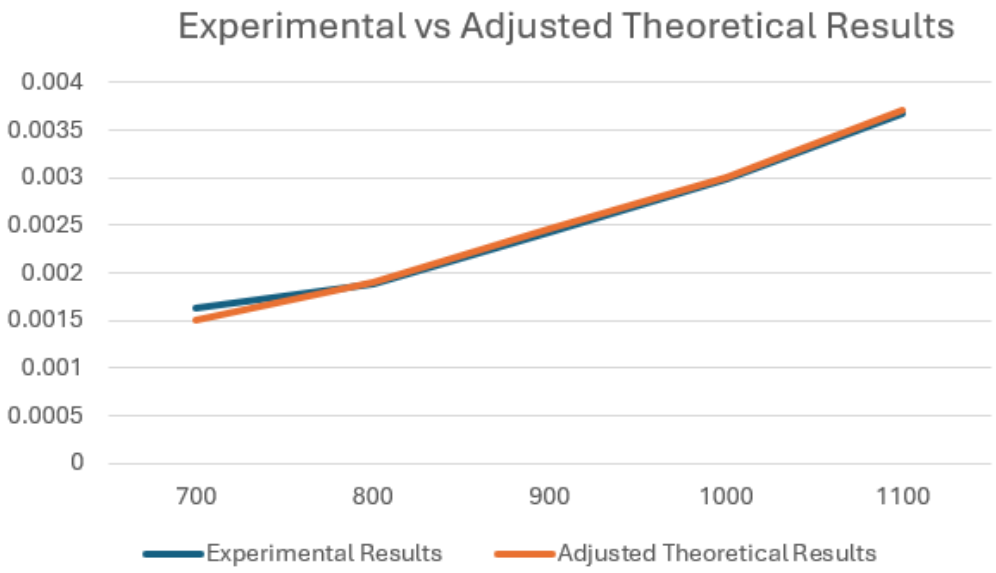
3. Time Complexity

The above function uses dynamic programming with a DP table, where each entry `dp[i][j]` checks if a subset sum `j` can be formed using the first `i` elements. After calculating the total sum `s`, we set the target as `s/2`. By iterating over each element and possibly sum up to `target`, the algorithm determines whether the array can be split into equal subsets. This approach runs in (O(ns), meeting the pseudo-polynomial time requirement due to `s` in the complexity.

4. Numerical Results

n	Experimental Results (seconds)	Theoretical Results	Scaling constant	Adjusted Theoretical Results (scaled)
700	0.00163483	2956800		0.001513693125
800	0.00188841	3729600		0.001909317464
900	0.00241723	4809600		0.002462208622
1000	0.00298992	5874000		0.003007113574
1100	0.0036786	7260000		0.003716657226
	0.002521798	4926000	0.000000000511936257	

5. Graph



6. Conclusion

The algorithm uses dynamic programming to determine if a set can be partitioned into two equal subsets by checking achievable subset sums up to s/2. With a time complexity of O(ns), where n is the number of elements and s is the total sum, it meets the pseudo-polynomial requirement. This approach efficiently builds solutions for each subset sum, making it suitable for moderate values of n and s. Thus, the algorithm successfully partitions the set, if possible, within the desired complexity.