CS 390 HW5 Problem 2

Aaryan Wadhwani

1 Small Numbers Case ($x_i \leq \epsilon T$)

Algorithm 1 Algorithm for Small Numbers

Input: A set of numbers $\{x_1, x_2, \ldots, x_n\}$ where $x_i \leq \epsilon T$, and target sum T
Output: A subset S with sum $S_{\text{sum}} \in [(1-\epsilon)T, (1+\epsilon)T]$

```
Initialize an empty subset S and set S_sum = 0
for each x_i in the list (in any order) do
    if S_sum + x_i ≤ T then
        Add x_i to S
        Update S_sum = S_sum + x_i
    end if
end for
return the subset S
```

1.1 Time Complexity

- The algorithm performs a single pass through the list, requiring $O(n)$ time.
- Total time complexity: $O(n)$, which is polynomial.

1.2 Correctness Proof

To demonstrate that the algorithm returns a subset S with sum $S_{\text{sum}} \in [(1-\epsilon)T, (1+\epsilon)T]$, we proceed as follows:

1. **Termination and Sum Bounds:** The algorithm adds numbers to S as long as $S_{\text{sum}} + x_i \leq T$. It stops adding numbers when either all numbers are included (so $S_{\text{sum}} \leq T$) or there exists an x_j not added because $S_{\text{sum}} + x_j > T$. Since $x_j \leq \epsilon T$, we have:
$$S_{\text{sum}} > T - x_j \geq T - \epsilon T = (1-\epsilon)T$$
Also, $S_{\text{sum}} \leq T$ holds because the algorithm only adds x_i if the sum remains at most T.

2. **Range Verification:** Combining the inequalities:

$$(1-\epsilon)T < S_{\text{sum}} \leq T$$

Since $T \leq (1+\epsilon)T$ (as $\epsilon > 0$), it follows that:

$$S_{\text{sum}} \in [(1-\epsilon)T, (1+\epsilon)T]$$

2 Big Numbers Case ($x_i > \epsilon T$)

Algorithm 2 Algorithm for Big Numbers

Input: A set of numbers $\{x_1, x_2, \ldots, x_n\}$ where $x_i > \epsilon T$, and target sum T
Output: A subset U with sum $S_U \in [(1-\epsilon)T, (1+\epsilon)T]$ or indication of no solution

Define $k_{\max} = \left\lfloor \frac{1+\epsilon}{\epsilon} \right\rfloor$
Enumerate all subsets $U \subseteq \{x_1, x_2, \ldots, x_n\}$ with $|U| \leq k_{\max}$
for each subset U **do**
    Compute $S_U = \sum_{x \in U} x$
    if $S_U \in [(1-\epsilon)T, (1+\epsilon)T]$ **then**
        return U
    end if
end for
If no such subset is found, indicate accordingly

2.1 Time Complexity

- Number of subsets: $\sum_{k=0}^{k_{\max}} \binom{n}{k} \leq \sum_{k=0}^{k_{\max}} n^k = O(n^{k_{\max}})$.
- Since $k_{\max} = O\left(\frac{1}{\epsilon}\right)$ and ϵ is a constant, this is $O(n^{O(1/\epsilon)})$.
- Computing S_U takes $O(n)$ per subset, so the total time is $O(n^{O(1/\epsilon)} \cdot n) = O(n^{O(1/\epsilon)})$.

2.2 Correctness Proof

To prove that the algorithm returns a subset U with sum $S_U \in [(1-\epsilon)T, (1+\epsilon)T]$, we proceed as follows:

1. **Bounding the Number of Elements:** Since each $x_i > \epsilon T$, the sum of a subset U with k elements satisfies $S_U > k \cdot \epsilon T$. To ensure $S_U \leq (1+\epsilon)T$, we require:

   $$k \cdot \epsilon T \leq (1+\epsilon)T \implies k \leq \frac{1+\epsilon}{\epsilon}$$

   Thus, $k_{\max} = \left\lfloor \frac{1+\epsilon}{\epsilon} \right\rfloor$ is the maximum number of elements that can yield a sum $\leq (1+\epsilon)T$. Enumerating all subsets with $|U| \leq k_{\max}$ ensures all relevant candidates are considered.

2. **Range Check:** The algorithm checks each U and returns the first with $S_U \in [(1-\epsilon)T, (1+\epsilon)T]$. If no such U exists, it indicates failure, aligning with the problem's assumption of a feasible solution.

3. **Completeness and Feasibility:**

- **Lower Bound:** For $S_U \geq (1-\epsilon)T$, a subset with $k \geq \frac{1-\epsilon}{\epsilon}$ elements could exceed $(1-\epsilon)T$ if $k \cdot \epsilon T > (1-\epsilon)T$. However, $k_{\max}$ limits the search to valid combinations, and the existence of a solution is assumed.
- **Upper Bound:** If $|U| > k_{\max}$, $S_U > (1+\epsilon)T$, which is outside the range. Thus, $k_{\max}$ is sufficient.
- **Existence:** If a subset with sum in $[(1-\epsilon)T, (1+\epsilon)T]$ exists and has at most $k_{\max}$ elements, the enumeration will identify it.

3 General Case (Both Big and Small Numbers)

Algorithm 3 Combined Algorithm for General Case

Input: A set of numbers $\{x_1, x_2, \ldots, x_n\}$ with some $x_i > \epsilon T$ and some $x_i \leq \epsilon T$, and target sum T

Output: A subset $U \cup V$ with sum $S_{\text{total}} \in [(1-\epsilon)T, (1+\epsilon)T]$ or indication of no solution

Partition the input into:
    $B = \{x_i \mid x_i > \epsilon T\}$ (big numbers)
    $S = \{x_i \mid x_i \leq \epsilon T\}$ (small numbers)
Define $k_{\max} = \left\lfloor \frac{1+\epsilon}{\epsilon} \right\rfloor$
Enumerate all subsets $U \subseteq B$ with $|U| \leq k_{\max}$
for each subset U **do**
    Compute $S_U = \sum_{x \in U} x$
    if $S_U > T$ **then**
        Skip this U
    else
        Initialize V as an empty subset and $\text{sum}_V = 0$
        for each x_i in S (in any order) **do**
            if $\text{sum}_V + x_i \leq T - S_U$ **then**
                Add x_i to V
                Update $\text{sum}_V = \text{sum}_V + x_i$
            end if
        end for
        Compute $S_{\text{total}} = S_U + \text{sum}_V$
        if $S_{\text{total}} \in [(1-\epsilon)T, (1+\epsilon)T]$ **then**
            return $U \cup V$
        end if
    end if
end for

3.1 Time Complexity

- Partitioning: $O(n)$.
- Enumerating U: $O(n^{k_{\max}}) = O(n^{O(1/\epsilon)})$ since $k_{\max} = O(1/\epsilon)$.
- Computing S_U: $O(n)$ per subset, total $O(n^{O(1/\epsilon)} \cdot n)$.
- Processing S per U: $O(n)$ per subset, total $O(n^{O(1/\epsilon)} \cdot n)$.
- Total: $O(n) + O(n^{O(1/\epsilon)} \cdot n) = O(n^{O(1/\epsilon)})$, polynomial for constant ϵ.

3.2 Correctness Proof

To prove that the algorithm returns a subset $U \cup V$ with sum $S_{\text{total}} = S_U + \text{sum}_V \in [(1 - \epsilon)T, (1 + \epsilon)T]$, we proceed as follows:

1. **Bounding Big Numbers (U):** Since $x_i > \epsilon T$ for $x_i \in B$, a subset U with k elements has $S_U > k \cdot \epsilon T$. To ensure $S_U \leq T$ (to proceed with adding small numbers), $k \leq \frac{1}{\epsilon}$. For the total sum to be at most $(1 + \epsilon)T$, $k \leq \frac{1+\epsilon}{\epsilon}$. Thus, $k_{\max} = \left\lfloor \frac{1+\epsilon}{\epsilon} \right\rfloor$ ensures all feasible U are considered.

2. **Selection of Small Numbers (V):** For a fixed U with $S_U \leq T$, the algorithm adds $x_i \in S$ to V if $\text{sum}_V + x_i \leq T - S_U$. When it stops, either all small numbers are included, or there exists $x_j \in S$ such that $\text{sum}_V + x_j > T - S_U$. Since $x_j \leq \epsilon T$:

   $$\text{sum}_V > (T - S_U) - x_j \geq (T - S_U) - \epsilon T$$

   Thus:
   $$S_{\text{total}} = S_U + \text{sum}_V > S_U + (T - S_U) - \epsilon T = T - \epsilon T = (1 - \epsilon)T$$

3. **Upper Bound Verification:** Since $\text{sum}_V \leq T - S_U$ and $S_U \leq T$:

   $$S_{\text{total}} = S_U + \text{sum}_V \leq S_U + (T - S_U) = T \leq (1 + \epsilon)T$$

4. **Range Satisfaction:** Combining the bounds:

   $$(1 - \epsilon)T < S_{\text{total}} \leq (1 + \epsilon)T$$

   Thus, $S_{\text{total}} \in [(1 - \epsilon)T, (1 + \epsilon)T]$.