HOMEWORK - 17

1. (a) Aggregate Method

* The tables doubles in size, when it runs out of space.
* If the original size is 1, after insertion it doubles to size 2, after one more insertion it doubles to size 4, and so on.
* After k doublings, the size is 2^k.

Pseudocode

```
Initialize table with capacity = 1
for i = 1 to n:
    if table is full:
        new_table = create new table with size 2 * current size
        copy elements from old table to new_table
        table = new_table
    insert element i into table
```

Let $k = \log(n+1) - 1$. The total cost for all operations is approximately $O(n \log n)$, and the amortized cost per insertion is $O(\log n)$. Therefore, the time complexity per insertion is $O(\log n)$, and the total time for n insertions is $O(n \log n)$.

(b) Accounting Method

Each insertion is charged a fixed cost of 2 units. When the table grows from size m to size 2m, we assign a credit of m units to cover the cost of copying the elements during

the doubling process. This ensures the cost of resizing is evenly distributed across all operations.

Pseudocode

```
Initialize table with capacity = 1
for i = 1 to n:
    if table is full:
        new_table = create new table with size 2 * current size
        copy elements from old table to new_table
        table = new_table
    insert element i into table
Initialize charge = 0
Initialize credit = 0
for i = 1 to n:
    charge += 2
    if table doubled in size from m to 2m:
        credit += m
```

The total charge across all insertions is $2n$, or $O(n)$. The total credit accumulated over the doublings is also $O(n)$. The amortized cost per insertion is $O(1)$, and the time for each insertion is $O(1)$. Hence, the total time for n insertions is $O(n)$.