1) (10 pts) ANL (Algorithm Analysis)

Consider storing a table with indexes 0 to N-1, where $N = k^2$, for some positive integer k, that starts with all entries equal to 0 and allows two types of operations: (1) adding some value to a particular index, and (2) querying the sum of all the values in the table from index 0 through index *m*, for any positive integer $m < N$. One way to implement a "table" to handle these two operations is to store two separate arrays, *groups*, of size k and *freq*, of size N. *freq stores* the current value of each index in the table. For the array *groups*, index i ($0 \le i < k$) stores the sum of the values in *freq* from index iN to index (i+1)N-1. (For example, if N = 25, then *groups[2]* stores the sum of the values of freq, from *freq[10]* through *freq[14]*, inclusive.

Determine, ***with proof***, the run-time of implementing operation (1) on this table using this storage mechanism and determine, ***with proof***, the run-time of implementing operation (2) on this table using this storage mechanism. (For example, if N = 100 and we had a query with *m* = 67, to get our answer we would add *groups[0]*, *groups[1]*, *groups[2]*, *groups[3]*, *groups[4]*, *groups[5]*, *freq[60]*, *freq[61]*, *freq[62]*, *freq[63]*, *freq[64]*, *freq[65]*, *freq[66]* and *freq[67]*. Notice that since the ranges 0-9, 10-19, 20-29, 30-39, 40-49, and 50-59 are fully covered in our query, we could just use the groups array for each of those sums. We only had to access the freq array for the individual elements in the 60s.)

Your answers should be Big-Oh answers in terms of N as defined above.

To add a value to a particular index in the table, we must do one update in each of our two arrays. For example, to add x to table index i, we would do these two updates:

```
freq[i] += x;
groups[i/k] += x;
```

Namely, we are redundantly storing our information in two places, so both places must be updated. This runs in O(1) time since each update is a simple statement/command.

A query has a different analysis since we are looking for the sum of items in the table from index 0 through some given index m, where m can range from 0 all the way to N-1. The key observation though is that we will never look at all items in freq for any query. If our query is to a "large value" of m, by adding multiple values in groups, we can do our work more quickly, adding k values at a time. In the worst case, we will add at most k values from the groups array. Notice that since the groups array entries represent table sums of k elements, when we have to add items from the freq array, we will never add more than k of them, since if we were to have added k of them, we could have just added one more value from the groups array. Thus, we do a maximum of k accesses to the groups array and a maximum of k-1 accesses to the freq array, for a total of O(k) time. Since the question asks to respond in terms of N, note that $k = \sqrt{N}$, so the run time of a query operation is $O(\sqrt{N})$

Grading: 1 pt for update answer, 3 pts for proof, 2 pts for query answer, 4 pts for proof. Latter proof should explain why no more than k accesses of the freq array are necessary to handle any query. Give partial for proofs as you see fit. There is no need for descriptions to be as long or detailed as the solution given above.