Dear Riley,

Thank you for reaching out to me and I hope you are doing well. I am writing to inform you that I have finished the task that you assigned to me. Attached to this email is the Python code I wrote to solve this problem. As well, I have provided an explanation for my code below, followed by the amortized complexity analyses for both the scenarios described in your email.

Within the starter code I was given, I implemented the provided `insert`, `search`, and `delete` methods.

The `insert` method, given a key k and value v, inserts the key and value into the hash table with capacity c in two steps. To start, it inserts the given key and value into the hash table. First, it hashes the key using the given hash function, whose `capacity` argument is left empty. I will denote this hashed value as $h(k)$. Then, the function checks `self.array` at index $h(k)$. If the address is unoccupied or if it has already been occupied by the key k, then a key-value pair `Node`(k, v) is inserted into the array at that address. Otherwise, if the address is occupied by another key, then using quadratic probing, the method continually checks the array at indices $h_i = h(k) + i^2 \pmod{c}$ in order, where i is a positive integer. The program stops when it finds an address that does not contain the key k is encountered, and inserts the key-value pair at the specified address. Note that this algortihm is guaranteed to terminate given that we know capacity of the hash table is less than half, although proving that is the case is quite lengthy. However, I am willing to elaborate if need be.

After a successful insertion, if the capacity of the hash table exceeds half, the method creates a new array with double the capacity of the current array, and iterates through the current array to reinsert the existing elements into the new array, following the steps outlined above. The only difference is when hashing each key, the hash function is called with the capacity parameter set to twice the current capacity.

That is the explanation for how `insert` works.

I hope that my explanations and analyses were satisfactory, and I look forward to hearing back from you soon. Thank you very much.

Ethan Hua