Hashing (10 points)

1. [2 points] A bit vector is simply an array of bits (0s and 1s). A bit vector of length m takes much less space than an array of m pointers. Describe how to use a bit vector to represent a dynamic set of distinct elements with no satellite data. Dictionary operations should run in $O(1)$ time.

> **Solution:** Start with a bit vector b which contains a 1 in position k if k is in the dynamic set, and a 0 otherwise. To search, we return true if $b[x] == 1$. To insert x, set $b[x] = 1$. To delete x, set $b[x] = 0$. Each of these takes $O(1)$ time.

2. [2 points] Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$?

> **Solution:** Under the assumption of simple uniform hashing, we will use linearity of expectation to compute this. Suppose that all the keys are totally ordered $\{k_1, \ldots, k_n\}$. Let X_i be the number of $\ell > k_i$ so that $h(\ell) = h(k_i)$. Note, that this is the same thing as $\sum_{j>i} \Pr(h(k_j) = h(k_i)) = \sum_{j>i} 1/m = (n-i)/m$. Then, by linearity of expectation, the number of collisions is the sum of the number of collisions for each possible smallest element in the collision. The expected number of collisions is $\sum_{i=1}^{n} \frac{n-i}{m} = \frac{n^2 - \frac{n(n+1)}{2}}{m} = \frac{n^2-n}{2m}$.

3. [2 points] Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

> **Solution:** Both kinds of searches become expected runtime of $\Theta(1+\lg(\alpha))$. Insertions and deletions stay $\Theta(1+\alpha)$ because the time to insert into or delete from a sorted list is linear.

4. [4 points] Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m-1))$.

> **Solution:** This is what the array will look like after each insertion when using linear probing:

										10
22										10
22									31	10
22				4					31	10
22				4	15				31	10
22				4	15	28			31	10
22				4	15	28	17		31	10
22	88			4	15	28	17		31	10
22	88			4	15	28	17	59	31	10

For quadratic probing, it will look identical until there is a collision on inserting the fifth element. Then, it is

22				4				15	31	10
22				4		28		15	31	10
22			17	4		28		15	31	10
22		88	17	4		28		15	31	10
22		88	17	4		28		15	31	10

Note that there is no way to insert the element 59 now, because the offsets coming from $c_1 = 1$ and $c_2 = 3$ can only be even, and an odd offset would be required to insert 59 because $59 \bmod 11 = 4$ and all the empty positions are at odd indexes.

For double hashing, it is the same as linear probing.