14.1. Let the number of possible keys be $U > nm$, where n is the number of elements in the hash table, and m is its size. Show that for any hash function, in the worst case, it is possible for all n elements to fall into same bucket of the hash table.

14.2. Let $h(k)$ be a random hash function. What is the mathematical expectation of the number of collisions (the number of pairs $(x, y), x \neq y$ such that $h(x) = h(y)$)?

14.3. We will resolve collisions using lists, but keep the lists in sorted order. How does this affect the worst case and average time complexity?

14.4. Add to the hash table the ability to iterate all its elements in the order in which they were added in $O(n)$.

14.5. Add the `merge` operation to the hash set, which combines two sets into one. Amortized running time $O(\log n)$.

14.6. See how the `Long.hashCode()` method works in Java. How can you generate many `Long`s with the same hash? Try to put them in `HashSet<Long>` and then put in another `HashSet <Long>` the same number of random `Long`s, look at how much runtime differs.

14.7. The same with `String.hashCode()`.

14.8. Add a `countUnique` operation to the dequeue, which returns the number of distinct items in the queue. Time $O(1)$.