Answers to Self-Study Questions

Test Yourself #1

- **constructor**: This method allocates the initial array, sets current to -1 and sets numItems to 0. This has nothing to do with the sequence size and is constant-time.
- **add** (to the end of the list): In the worst case, the array was full and you have to allocate a new, larger array, and copy all items. In this case the number of operations is proportional to the size of the list. If the array is not full, this is a constant-time operation (because all you have to do is copy one value into the array and increment numItems).
- **add** (at a given position in the list): As for the other version of add, if the array is full, time proportional to the size of the list is required to copy the values from the old array to the new array. However, even if the array is not full, this version of add can require time proportional to the size of the list. This is because, when adding at position k, all of the items in positions k to the end must be moved over. In the worst case (when the new item is added at the beginning of the list), this requires moving all items over, and that takes time proportional to the number of items in the list.
- **isEmpty**: This method simply returns the result of comparing numItems with 0; this is a constant=time operation.
- **contains**: This method involves looking at each item in the list in turn to see if it is equal to the given item. In the worst case (the given item is at the end of the list or is not in the list at all), this takes time proportional to the size of the list.
- **get**: This method checks for a bad position and either throws an exception or returns the value in the given position in the array. In either case it is independent of the size of the list and so it is a constant-time operation.

Test Yourself #2

Question 1: The problem size is the number of people in the room.

Question 2: Assume there are N people in the room. In algorithm 1 you always ask 1 question. In algorithm 2, the worst case is if no one has your birthday. Here you have to ask every person to figure this out. This is N questions. In algorithm 3, the worst case is the same as algorithm 2. The number of questions is 1 + 2 + 3 + ... + N-1 + N. We showed before that this sum is N(N+1)/2.

Question 3: Given the number of questions you can see that algorithm 1 is constant time, algorithm 2 is linear time, and algorithm 3 is quadratic time in the problem size.

Test Yourself #3

1. The first loop is O(N) and the second loop is O(M). Since you don't know which is bigger, you say this is O(N+M). This can also be written as O(max(N,M)). In the case where the second loop goes to N instead of M the complexity is O(N). You can see this from either expression above. O(N+M) becomes O(2N) and when you drop the constant it is O(N). O(max(N,M)) becomes O(max(N,N)) which is O(N).
2. The first set of nested loops is $O(N^2)$ and the second loop is O(N). This is $O(\max(N^2,N))$ which is $O(N^2)$.
3. This is very similar to our earlier example of a nested loop where the number of iterations of the inner loop depends on the value of the index of the outer loop. The only difference is that in this example the inner-loop index is counting down from N to i+1. It is still the case that the inner loop executes N times, then N-1, then N-2, etc, so the total number of times the innermost "sequence of statements" execites is $O(N^2)$.

Test Yourself #4

1. Each call to `f(j)` is O(1). The loop executes N times so it is N x O(1) or O(N).

2. The first time the loop executes `j` is 0 and `g(0)` takes "no operations". The next time `j` is 1 and `g(1)` takes 1 operation. The last time the loop executes `j` is N-1 and `g(N-1)` takes N-1 operations. The total work is the sum of the first N-1 numbers and is $O(N^2)$.
3. Each time through the loop `g(k)` takes k operations and the loop executes N times. Since you don't know the relative size of k and N, the overall complexity is O(N x k).