2.1

1. A relatively small array, with a size of eight for example, would easily fall below half full following a short series of deletions. After only a few insertions, it would need to be expanded again, and a few deletions would have it reduced again.

2. The underflow could require that the array:

* Has considerable size, such as 1,000
* Is highly under populated, say, only 10% full
* Decreases with room to spare, for example, leaves array at half full

Alternatively, the underflow method could be called only when the process using the array is finished with a series of operations, or wants to clean up or free memory.

2.2

The makeList method is O(N^2) because trimToSize() is O(N). Since trimToSize() is inside makeList()'s O(N) loop, the result is O(N^2).

2.3

a) The first call passes 50 into the fibonnaci method. It calls fibonnaci again with N-1 and N-2, and since N-1 would decrement slower, we follow that route. The second call passes in 49, third is 48, and on the 50th call, the number passed in is 1, which returns 1, instead of continuing the recursion. Therefore, 50 will be the most calls stored in the stack.

b) No, since the stack can have more than a hundred calls. This was also confirmed with a test run, which didn’t encounter errors.

c) No, since fib() has an exponential Big-O, it would take a long time to complete. This was confirmed in the test for part B.

2.4

a) It’s saved so the loop knows how many elements to remove.

b) O(N^2) because remove() for an ArrayList is O(N), and the loop in removeFirstHalf() executes remove() N/2 times, resulting in O(N^2)

c) O(N) because remove() on the first element in a LinkedList is constant time, and removeFirstHalf()’s loop executes remove() N/2 times.

d) No, since its operation would not be too different from the provided loop.

2.5

Converted to postfix:

|  |  |
| --- | --- |
| Stack | Output |
| ( |  |
| ( | 600 |
| ( - | 600 |
| ( - | 600 203 |
|  | 600 203 - |
| \* ( | 600 203 - |
| \* ( | 600 203 - 143 |
| \* ( + | 600 203 – 143 |
| \* ( + | 600 203 – 143 21 |
| \* | 600 203 – 143 21 + |
| \* | 600 203 – 143 21 + |
| \* | 600 203 – 143 21 + |
| \* | 600 203 – 143 21 + |
| / | 600 203 – 143 21 + \* |
| / ( | 600 203 – 143 21 + \* |
| / ( | 600 203 – 143 21 + \* 87 |
| / ( + | 600 203 – 143 21 + \* 87 |
| / ( + | 600 203 – 143 21 + \* 87 17 |
| / | 600 203 – 143 21 + \* 87 17 + / |