1. To test insert and deleteMin. I first create another list to contain the same values with heap that we are testing. Then sort the answer using method java provided to match insert method, where it finds the minimum number to be the root. Then using deleteMin from heap add to another list to check if the deleteMin delete the smallest value from the heap. If the values match each values in the sorted list then both insert and deleteMin method works. Same testing concept with buildQueue method.
2. buildHeap: O(N), isEmpty: O(1), size: O(1), insert: O(log N), findMin: O(1), deletemin: O(log N)
3.
    a. O(log n)
    b. O(n^3)
    c. O(n)
    d. O(n^2)
4.
    a. maxDiff = 0

        for i from 0 to array.size - 1

            for j from i + 1 to array.size

                if array[j] – array[i] > maxDiff

                    maxDiff = array[j] – array[i]

    b. maxDiff = 0

        min = array[0]

        for i from 1 to array.size

            if array[i] – min > maxDiff

                maxDiff = array[i] – min

            if array[i] < min

                min = array[i]

    c. It can't happen, but if the array is sorted then we can just find the first one which is the smallest and last one which is the largest, subtract them to get the largest difference.

5.

a. O(N^2). When the array is empty, the first four elements are inserted in time O(1). At the fifth insert, the time needed is one insert (O(1)) plus the copy (O(5)). The algorithm works exactly like this until it finished inserting N items. In mathematical formula:

$$T(n) = \left(\frac{n}{5}\right) * 4 + (5 + 10 + 15 + \cdots + n) = \frac{4n}{5} + \frac{(5+n) * \frac{n}{5}}{2} = \frac{4n}{5} + \frac{5n + n^2}{10}$$

b. O(N). Assume we have N elements in the array and the array is full. The time to copy the array is O(N). Since before this copy, the algorithm performed approximately N/2 operations of insert using O(N) time, we can amortize this expensive O(N) operation to the previous insert operations. So the average time of every operation is $1 + \frac{N}{\frac{N}{2}}$, which is 3. So every insert has average time of constant. Therefore, inserting n items uses time of O(N).

c. O(N), same reason with the second one

6. Using Floyd's method to implement the buildQueue method in O(N) run time.