M3:Post Lab

After you have successfully completed your implementation of the min-heap operations and generated the tick count curve as described in the lab instructions, answer the following questions.

First, for each of the four methods below, describe briefly (in three sentences or less) how it works. In addition, answer the following questions: Did the method use any instance variables of the class, and if so, which ones? Did you create any helper methods, and if so, what do they do?

The four methods that you should describe are:

1. `public Decreaser<T> insert(T thing){}`
2. `void decrease(int loc){}`
3. `public T extractMin(){}`
4. `private void heapify(int where){}`

Now answer the following additional questions:

5. Include the tick-count graph obtained from the `HeapTimer` experiment (as described in the lab instructions) as a figure in your writeup. Does the curve have the shape suggested in the instructions?

   Tip: Your graph may look linear at first glance, so you may not be able to tell right away whether you've correctly placed tick calls to show the correct complexity, or whether your graph is linear instead. Here are two ways to check whether your timing data is proportional to n or something greater (the second method specifically tests against $n \log n$):

   (a) Take the first two time points you have and extrapolate a line from the points, generating a new point for each value of n in your graph. Graph the new line and your timing data on the same graph, and you should see that your timing data graph curves above the line as n increases. This indicates a super-linear running time.

   (b) Generate a column in your time-data spreadsheet that calculates $n \log n$ for each value of n. Then generate a column for *measured_time* / $n \log n$, where *measured_time* is your ticks value for a given n. The values in this column should be constant (perhaps with slight variance). If so, then your ticks output is some constant multiple of $n \log n$, which is what you want to see. If the values in this column are steadily increasing, then your ticks output is likely asymptotically greater than $n \log n$, which indicates that your code is slower than it should be assuming your tick calls are placed correctly.

6. What is the running time of calling `extractMin` n times on a heap of size n? (Note that *size* refers here to the number of elements in the heap rather than the total *capacity* of the heap.) Give an asymptotic upper bound (O) on the running time as a function of n, and justify this bound.

7. Briefly describe what the heap property is and how it is maintained in the code during the `insert` and `extractMin` methods.

8. Could one side of a min-heap ever become very tall compared to the other side? To be precise, could the maximum height of any node in the left subtree of a min-heap ever differ from the max height of any node in the right subtree by more than (say) 5? If so, give a sequence of insertions that would produce such a min-heap; if not, explain why.

9. Name one advantage and one disadvantage of storing data in a min-first heap compared to storing it in an unordered list.