1. public Decreaser<T> insert(T thing){}

- What it does?
    - This method add a new node(which is type T) in the MinHeap class and make sure the node will be settled in the right position of the Heap. When you set 'thing' parameter and call this method and then this method will create a 'Decreaser' object which holds the 'thing' value and make it to the right position.
- Does it use any instance variables of the class?
    - Yes, it uses two instance variables, array variable and size variable.
    - After creation of the new node, insert method put newly create node in the array. So it has to know what is array variable and its size.
- Does it have any helper methods?
    - No.

2. void decrease(int loc){}

- What it does?
    - This method will move the node from the 'loc' position to the right position in the MinHeap class. If the pointed node's value is smaller than its parent's and then swap the position. And then repeat this process until the top node or there are no more swaps.
- Does it use any instance variables of the class?
    - Yes, it uses one instance variables, array variable.
    - Because this method needs the handle of Decreaser objects.
- Does it have any helper methods?
    - Yes. `void swapDecreaserNode(Decreaser<T> a, Decreaser<T> b)`
    - As you guess from the name of the helper method, it will swap the two nodes. The important point of this implementation is to make sure swap the 'loc' variable of 'Decreaser' instance.

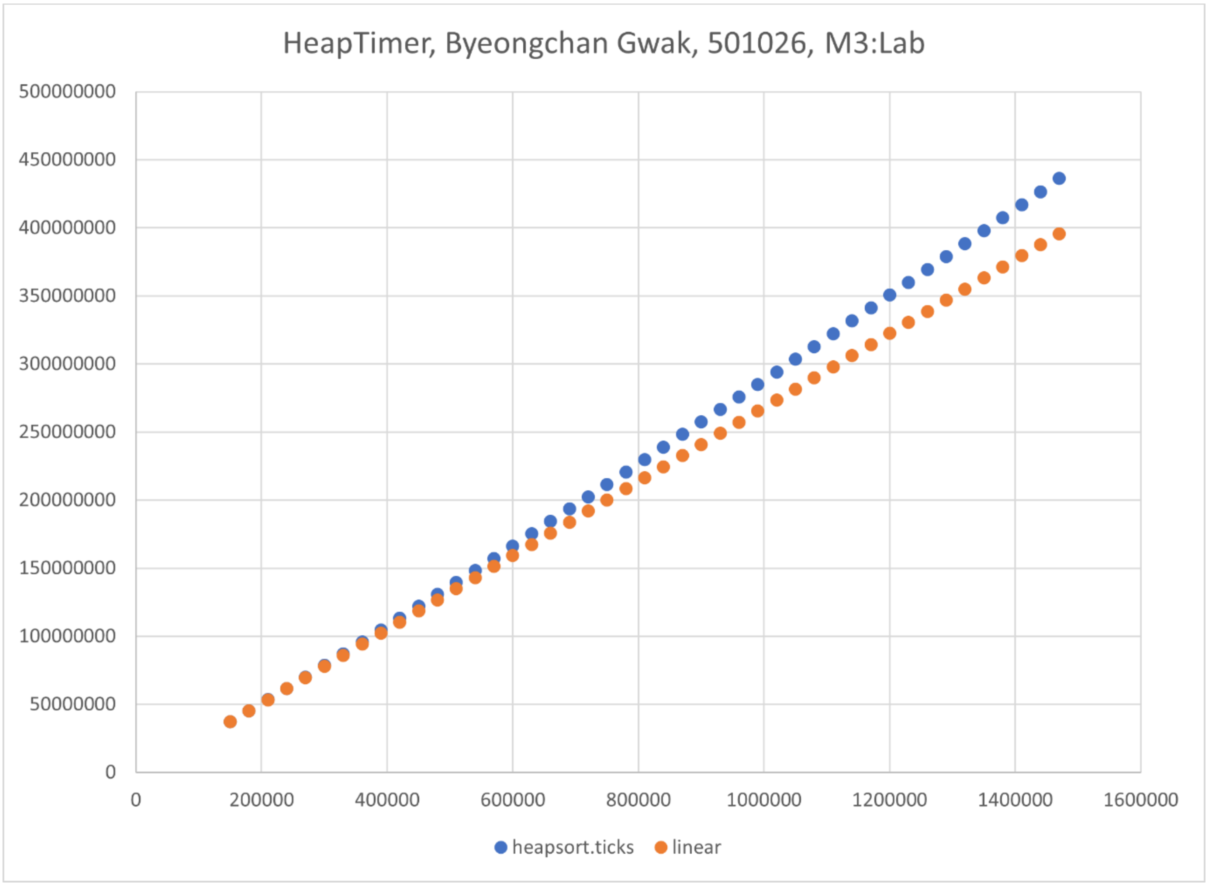
3. public T extractMin(){}

- What it does?
  - Extract a top node of the MinHeap class and return the node. This method also makes sure the MinHeap class satisfy heap structure after extracting. To do so, this method will call heapify() inside the code.
- Does it use any instance variables of the class?
  - Yes, it uses two instance variables, array variable and size variable.
  - After extracting the top node, this method should move the last node into to top position and subtract the array size by 1.
- Does it have any helper methods?
  - Yes. `private void heapify(int where) {`
  - As you guess from the name of the helper method, it will heapify the nodes inside the 'MinHeap' class. It will start from the top of the MinHeap class and move the top node into the right position.

4. private void heapify(int where){}

- What it does?
  - This will heapify the nodes inside the MinHeap class. It's something like bubble-down the node. If the node's value is bigger than the children's minimum value and then swap the two node. Repeat this process until the bottom of the tree or there is no swap.
- Does it use any instance variables of the class?
  - Yes, it uses two instance variables, array variable and size variable.
  - To swap the node, you need a handle of each node. And also I use a size variable to make sure not to access over the size of the array.
- Does it have any helper methods?
  - Yes. `void swapDecreaserNode(Decreaser<T> a, Decreaser<T> b)`
  - As you guess from the name of the helper method, it will swap the two nodes. The important point of this implementation is to make sure swap the 'loc' variable of 'Decreaser' instance.

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?



- Does the curve have the shape suggested in the instructions?
  - Yes, compare to the linear graph, I assume that it draws $n \cdot \log n$ graph.

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.

- Getting a time complexity of extractMin method on a heap of size n : $\Theta(\log n)$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + k$$

depth	size	#nodes	Work per node	Total work
0	n	1	k	k
1	$\frac{n}{2^1}$	2^1	$\frac{k}{2^1}$	k
2	$\frac{n}{2^2}$	2^2	$\frac{k}{2^2}$	k
…	…	…	…	…
i	$\frac{n}{2^i}$	2^i	$\frac{k}{2^i}$	k
$\log_2 n$	1	n	d	d

- Getting a time complexity of calling a extractMin method n times : $\Theta(n \cdot \log n)$
- Hence, the time complexity of calling a extractMin method n times : $O(n \cdot \log n)$

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

- Heap property
  - Shape property : It should be a complete binary.
  - Order property : Any parent data < the data in it's children
- insert method()
  - Inside this method, it will call decrease() method which can bubble-up the node into the top of the Heap. So whenever insert() method is called, it will also call decrease() method to make sure the heap structure.
- extractMin()
  - Inside this method, it will call heapify() method which can bubble-down the node until the bottom of the tree. So whenever extractMin() method is called, it will also call heapify() method to maintain the heap structure.

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.

- It can’t happen. Because min-heap class is a complete binary tree. In a complete binary tree, the tree on the left and the tree on the right can differ by at most one level. We can think of it by dividing it into inserting and extracting phase.
- Let’s say that there is a min-heap data structure which follow the ‘Shape property’. When insert a new node to the min-heap, we put the new node at the left end of the heap tree structure. Therefore the ‘Shape property’ doesn’t break because it is still complete binary tree. After that, the new node will find its right position doing swapping. Swapping also doesn’t break the ‘Shape property’. The complete binary tree structure cannot be break during insertion.
- When extracting a node to the min-heap, we delete the top node and replace the hole with the left end node of the tree. Still, ‘Shape property’ doesn’t break because it is still complete binary tree. After that, last node will find its right position doing swapping. Swapping also doesn’t break the ‘Shape property’. The complete binary tree structure cannot be break during extraction.
- Therefore, ‘one side of a min-heap ever become very tall compared to the other side’ isn’t true.

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

- Advantage
  - When extract min value from the MinHeap, the time complexity is $O(\log_2 n)$, while unordered list is $O(n)$
- Disadvantage
  - When inserting a new node, time complexity of MinHeap is $O(\log_2 n)$, while unordered list is $O(1)$