Q1:

Given a max-heap of *n* values and you have a function `DecreaseValue(int x, int y)` which will take the value at index *x*, and decrease its value by subtracting the value *y* to it. Now that value has been changed, you might not have a legal max-heap any longer, so you need to then do the **minimum work** necessary with the **best** running time you could apply on, to repair the max-heap.

Supposedly a function `RepairMaxHeap` does all that. What is the worst-case running time of `RepairMaxHeap`? Justify convincingly why your answer is correct and express your answer in Big-O notation.

- If in a max-heap of n values we try to change the value of element x by decreasing it by y, then we might have a illegal max-heap. That would happen if the element x-y will have a smaller value than its children. In this case, we should "sink" the element with one of its children. We can do some replacement by swapping the value x-y with its larger children. If that violation still continues, we do the same thing again, until we have a legal max-heap.
- **The time complexity would be O(logn),** as in each step, we always consider one of the two possibilities (children swapping). So, the number of nodes is always divided by two making the time logarithmic.