3) This algorithm could use recursion to consider the left and right children. Because every child of a min heap is larger than its parent, an algorithm proportional to O(k) would need to stop iterating on a subtree once the value of the nodes become higher than x. Iterating through these values that are deeper in the tree become unnecessary, as we already know that we will not count any node in that subtree, so we can move on to the subtree directly "to the left" of the last node that was < x.

Pseudo Code:

```
let numLessThan = 0
let index = 0
def numLessThanX(minHeap, x, index):
    Base cases
    if the index > minHeap.length -> Break, to control the num of times we loop
    if minHeap[index] >= x -> Break, as we know from the definition of a minheap
    that all children of this particular subtree will also break
    Recursive step
     if the two base cases are not satisfied
      numLessThan++
       Call numLessThan on left subtree -> Change value of index
       Call numLessThan on right subtree -> Change value of index
                                                  ] -> Increment by the value of the implementation of array
def atLeastKItems(minHeap, k, x):
        numLessThanX(minHeap, x, index)
        return numLessThan >= k
```