(5) **(Reducing kth-smallest to median-finding)** (15 points) If we have an algorithm that finds the kth-smallest element of an n element set, we can obtain an algorithm for finding the median element simply by calling the kth-smallest algorithm with $k = \lceil (n+1)/2 \rceil$. This question asks you to do the converse.

Given an algorithm that finds the median element of an n element set in $\Theta(n)$ time, design an algorithm that finds the kth-smallest element for arbitrary k in $\Theta(n)$ time using the median-finding algorithm. Be sure to analyze the running time of your algorithm.

(Note: Do *not* invoke the linear-time kth-smallest algorithm presented in class and the book. You must reduce the problem of finding the kth-smallest element to the problem of finding the median.)

Note that Problems (3)(c) and (4)(e) are *bonus* questions, and are *not* required.

First we use the median element finding algorithm to find the median position. From there we know the index of the median will be $med = \lceil n+1/2 \rceil$ $\Theta(n)$

Then, determine if $med = k$ or $med > k$ or $med < k$ $\Theta(1)$

Next, run through the array and place in a new array all less than the median if $med > k$ or greater than the median if $med < k$ $\Theta(n)$ time
call this new array A'[ ] $\Theta(n)$ extra space

Next, min-heapify the additional array $\Theta(\log n)$

If $med > k$ remove the root of the heap and re-heapify $\Theta(\log n)$
– Do this k times to find the k^{th}-smallest element $\Theta(k)$ $\Big\} \Theta(k \log n)$

If $med < k$ remove the root of the heap and re-heapify $\Theta(\log n)$
– Do this $k - (A.len - A'.len)$ times $\Theta(k - (A.len - A'.len))$ $\Big\} \Theta(k - (A.len - A'.len) \cdot \log n)$
(the first elements up to and including the median haven't been included in this heap so we need to account for them)

In both of these cases we know k and $k - (A.len - A'.len)$ will be smaller than n since the median splits the original list into fractions of the list so both will become $\Theta(\log n)$

After removing the necessary number of elements from the heap we have the kth-smallest element.

We know this works because we are using the median to split the list then using heapsort to heapify the appropriate side. From there we can remove the smallest element from the heap (the root) the proper number of times to get the element we were looking for.

Time Analysis

```
Find median                                                    Θ(n)  } Θ(n)
If k == median : return median                                 Θ(1)  }
else if k < median DO
    create array and fill with elements < median  min A[]      Θ(n)
    min-heapify minA[]                                         Θ(logn)
    for i := 0 to k DO                                         Θ(k)     }
        s := root.remove                                       Θ(1)     } Θ(klogn)
        re-heapify                                             Θ(logn)  }
    end
end
else if k > median DO
    max A[] = array filled with elements > median              Θ(n)
    min-heapify maxA[]                                         Θ(logn)
    diff = A.length - max A.length
    for i := 0 to k - diff DO                                  Θ(k-diff) }
        s := root.remove                                       Θ(1)      } Θ((k-diff)·logn)
```

```
diff = A.length - max A.length
for i := 0 to k - diff DO
        s := root.remove
        re-heapify
    end
end
```

$\Theta(k - diff)$
$\Theta(1)$
$\Theta(\log n)$

$\Theta((k-diff) \cdot \log n)$

$\Theta(n + \cancel{k}\log n + (\cancel{k - diff})\log n) \rightarrow \Theta(n + \cancel{\log n} + \cancel{\log n}) \rightarrow \Theta(n)$

smaller than n