1. (10) Bucket Sort 8.4-2 (9.4-2 old book).
Worst case running time of Bucket sort is when all elements are placed in the same bucket. The resulting Insertion Sort then runs in $O(n^2)$ so the worst-case running time of Bucket sort is:

$$T(n) = \Theta(n) + O(n^2) + \Theta(n) = O(n^2)$$

To improve the worst-case running time of Bucket Sort we could replace the Insertion Sort with a better comparison sort that has $O(n \log n)$ worst-case running time.

2. (10) Find Median 9.3-8 (10.3-8 old book)
Idea: Since the arrays are already sorted we know how to find median in $O(1)$ time. Assume that $n = 2^k$.
- First find two medians of the original arrays $X$ and $Y$. If $X[n/2] < Y[n/2]$ then find the median of the following subarrays $X[n/2 \ldots n]$ and $Y[1 \ldots n/2]$.
- If $X[n/2] > Y[n/2]$ then find the median of the following subarrays $X[1 \ldots n/2]$ and $Y[n/2 \ldots n]$.
- If $X[n/2] = Y[n/2]$ then we are done.

Write it down as a recursive algorithm FIND-MEDIANS($X[p_1 \ldots q_1]$; $Y[p_1 \ldots q_1]$). The recurrence describing the algorithm is $T(n) = 2T(n/2) + O(1)$. There are $\log n$ levels of the recursion tree, each takes $O(1)$. The total running time is $O(n \log n)$.

3. (15) Quantiles 9.3-6 (10.3-6 old book)
Example of an quantile: If $k = 2$ we have 1 element which divides the sorted set into two equal-sized parts. The array is initially not sorted. Assume for simplicity that $n = 2^m$ and $n = k^l$. Test the following idea of the algorithm on an example for $k = 16$ and $m = 4$:
- Find the median on set of $n$ elements in $O(n)$ time, considering the linear time algorithm for finding median, which we did in class. Partition the array around median, which also takes $O(n)$ time.
- Find the two medians of the sets of $n/2$ elements, each in $O(n/2)$ time.
- etc.

Suppose that $T(n)$ is the running time of the algorithm for finding median and doing the partitioning. The total running time of the algorithm is:

$$T(n) + 2T\left(\frac{n}{2}\right) + 2^2T\left(\frac{n}{4}\right) + \ldots + 2^{m-1}T\left(\frac{n}{2^{m-1}}\right) = \sum_{i=0}^{m-1} 2^iT\left(\frac{n}{2^i}\right)$$

$$O(n) + 2O\left(\frac{n}{2}\right) + 2^2O\left(\frac{n}{4}\right) + \ldots + 2^{m-1}O\left(\frac{n}{2^{m-1}}\right) = mO(n) = O(n \log k)$$

Look up the definition of the properties. Heap property is not strong enough to be able to print out the nodes in sorted order in $O(n)$ time. Otherwise we would have a $O(n)$ algorithm for sorting, because building heap takes $O(n)$ time. We know that comparison based sort takes at least $\Omega(n \log n)$ time.

5. (10) Bin Search Tree 12.2-7 (13.2-4)
Call to TREE-MINIMUM followed by $n - 1$ calls to TREE-SUCCESSOR performs exactly the same operations as INORDER tree walk. INORDER tree walk prints tree minimum first and then the tree successor the is the next node in the sorted order. The algorithm takes $\Theta(n)$ time because takes at least $\Omega(n)$ to do $n$ procedure calls. The algorithm traverses each of the $n - 1$ edge at most twice which takes $O(n)$ time. The only time the algorithm goes down an edge is during the call to TREE-MINIMUM (with the tree successor). The only time the algorithm goes up an edge, is during the while loop on TREE-SUCCESSOR.