CSE 5311 Homework Assignment 4 (Fall 2019)

Due date: 9/30 Monday (Type, print, and hand-in in class)

(1) Exercise 8.2-1 on Page 196 [10 points]
Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array \( A = \{6,0,2,0,1,3,4,6,1,3,2\} \).

Answer:

This is the array we sort: \( A: \{6 \ 0 \ 2 \ 0 \ 1 \ 3 \ 4 \ 6 \ 1 \ 3 \ 2\} \)

We build an array of counts: \( C: \{2 \ 2 \ 2 \ 1 \ 0 \ 2\} \)

The number of elements before each \( j \) of array \( A \):

\( C: \{2 \ 4 \ 6 \ 8 \ 9 \ 9 \ 11\} \)

Then we start iterating:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) Exercise 8.2-3 on Page 196 [20 points]
Suppose that we were to rewrite the for loop header in line 10 of the COUNTING-SORT as

\[
10 \ \textbf{for} \ j = 1 \ \textbf{to} \ A.length
\]

Show that the algorithm still works properly. Is the modified algorithm stable?
Answer:
Notice that the correctness argument in the text does not depend on the order in which A is processed. The algorithm is correct no matter what order is used. But the modified algorithm is not stable. As before, in the final for loop an element equal to one taken from A earlier is placed before the earlier one (i.e., at a lower index position) in the output array B. The original algorithm was stable because an element taken from A later started out with a lower index than one taken earlier. But in the modified algorithm, an element taken from A later started out with a higher index than one taken earlier. In particular, the algorithm still places the elements with value k in positions C[k-1]+1 through C[k], but in the reverse order of their appearance in A.

(3) Exercise 8.3-4 on Page 200 [10 points]
Show how to sort n integers in the range 0 to \(n^3 - 1\) in O(n) time.

Answer:
Treat the numbers as 3-digit numbers in radix n. Each digit ranges from 0 to \(n-1\). Sort these 3-digit numbers with radix sort. There are 3 calls to counting sort, each taking \(\Theta(n + n) = \Theta(n)\) time, so that the total time is \(\Theta(n)\).

(4) Exercise 9.1-1 on Page 215 [20 points]
Show that the second smallest of \(n\) elements can be found with \(n + \lceil \lg n \rceil - 2\) comparisons in the worst case. (Hint: Also find the smallest element.)

Answer:
The smallest of \(n\) numbers can be found with \(n - 1\) comparisons by conducting a tournament as follows: Compare all the numbers in pairs. Only the smaller of each pair could possibly be the smallest of all \(n\), so the problem has been reduced to that of finding the smallest of \([n/2]\) numbers. Compare those numbers in pairs, and so on, until there is just one number left, which is the answer.

To see that this algorithm does exactly \(n - 1\) comparisons, notice that each number except the smallest loses exactly once. To show this more formally, draw a binary tree of the comparisons the algorithm does. The \(n\) numbers are the leaves, and each number that came out smaller in a comparison is the parent of the two numbers that were compared. Each non-leaf node of the tree represents a comparison, and there are \(n - 1\) internal nodes in an \(n\)-leaf full binary tree (see Exercise (B.5-3)), so exactly \(n - 1\) comparisons are made.

In the search for the smallest number, the second smallest number must have come out smallest in every comparison made with it until it was eventually compared with the smallest. So the second smallest is among the elements that were compared with the smallest during the tournament. To find it, conduct another tournament (as above) to find the smallest of these numbers. At most \(\lceil \lg n \rceil\) (the height of the tree of comparisons) elements were compared with the smallest, so finding the smallest of these takes \(\lceil \lg n \rceil - 1\) comparisons in the worst case.

The total number of comparisons made in the two tournaments was \(n - 1 + \lceil \lg n \rceil - 1 = n + \lceil \lg n \rceil - 2\) in the worst case.
(5) Exercise 9.3.1 on Page 223 [20 points]
In the algorithm SELECT, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that SELECT does not run in linear time if groups of 3 are used.

**Answer:**

For groups of 7, the algorithm still works in linear time. The number of elements greater than x (and similarly, the number less than x) is at least
\[ 4 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2 \right) \geq \frac{2n}{7} - 8 \]
and the recurrence becomes
\[ T(n) \leq T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n) \]
which can be shown to be \( O(n) \) by substitution, as for the groups of 5 case in the text.

For groups of 3, however, the algorithm no longer works in linear time. The number of elements greater than x, and the number of elements less than x, is at least
\[ 2 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil - 2 \right) \geq \frac{n}{3} - 4 \]
and the recurrence becomes
\[ T(n) \leq T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n) \]
which does not have a linear solution.

We can prove that the worst-case time for groups of 3 is \( \Omega(n \log n) \). We do so by deriving a recurrence for a particular case that takes \( \Omega(n \log n) \) time.

In counting up the number of elements greater than x (and similarly, the number less than x), consider the particular case in which there are exactly \( \left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil \) groups with medians \( \geq x \) and in which the “leftover” group does contribute 2 elements greater than x. Then the number of elements greater than x is exactly
\[ 2 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil - 1 \right) + 1 \] (the -1 discounts x’s group, as usual, and the +1 is contributed by x’s group) = \[ 2[n/6]-1, \] and the recursive step for elements \( \leq x \) has \( n-(2[n/6]-1) \geq n-(2(n/6+1)-1) = 2n/3-1 \) elements. Observe also that the \( O(n) \) term in the recurrence is really \( \Theta(n) \), since the partitioning in step 4 takes \( \Theta(n) \) (not just \( O(n) \)) time. Thus, we get the recurrence
\[ T(n) \geq T(\lceil n/3 \rceil) + T(2n/3-1) + \Theta(n) \geq T(n/3) + T(2n/3-1) + \Theta(n), \]
from which you can show that \( T(n) \geq cn \log n \) by substitution. You can also see that \( T(n) \) is nonlinear by noticing that each level of the recursion tree sums to n. [In fact, any odd group size 5 works in linear time.]
Let \( X[1..n] \) and \( Y[1..n] \) be two arrays, each containing \( n \) numbers already in sorted order. Give an \( O(\log n) \) time algorithm to find the median of all \( 2n \) elements in arrays \( X \) and \( Y \).

**Answer:**

Let’s start out by supposing that the median (the lower median, since we know we have an even number of elements) is in \( X \). Let’s call the median value \( m \), and let’s suppose that it is in \( X[k] \). Then \( k \) elements of \( X \) are less than or equal to \( m \) and \( n - k \) elements of \( X \) are greater than or equal to \( m \). We know that in the two arrays combined, there must be \( n \) elements less than or equal to \( m \) and \( n \) elements greater than or equal to \( m \), and so there must be \( n - k \) elements of \( Y \) that are less than or equal to \( m \) and \( n - (n - k) = k \) elements of \( Y \) that are greater than or equal to \( m \).

Thus, we can check that \( X[k] \) is the lower median by checking whether \( Y[n-k] \leq X[k] \leq Y[n-k+1] \). A boundary case occurs for \( k = n \). Then \( n - k = 0 \), and there is no array entry \( Y[0] \); we only need to check that \( X[n] \leq Y[1] \).

Now, if the median is in \( X \) but is not in \( X[k] \), then the above condition will not hold. If the median is in \( X[k'] \), where \( k' < k \), then \( X[k] \) is above the median, and \( Y[n-k+1] < X[k] \). Conversely, if the median is in \( X[k'] \), where \( k' > k \), then \( X[k] \) is below the median, and \( X[k] < Y[n-k] \).

Thus, we can use a binary search to determine whether there is an \( X[k] \) such that either \( k < n \) and \( Y[n-k] \leq X[k] \leq Y[n-k+1] \) or \( k = n \) and \( X[k] \leq Y[n-k+1] \); if we find such an \( X[k] \), then it is the median. Otherwise, we know that the median is in \( Y \), and we use a binary search to find a \( Y[k] \) such that either \( k < n \) and \( X[n-k] \leq Y[k] \leq X[n-k+1] \) or \( k = n \) and \( Y[k] \leq X[n-k+1] \); such a \( Y[k] \) is the median. Since each binary search takes \( O(\log n) \) time, we spend a total of \( O(\log n) \) time.

Here is how we write the algorithm in pseudocode:

**TWO-ARRAY-MEDIAN(\( X, Y \))**

\[
\begin{align*}
n &\leftarrow \text{length}[X] \triangleright n \text{ also equals length}[Y] \\
\text{median} &\leftarrow \text{FIND-MEDIAN}(X, Y, n, 1, n) \\
\text{if} \quad \text{median} = \text{NOT-FOUND} &\quad \text{then} \quad \text{median} \leftarrow \text{FIND-MEDIAN}(Y, X, n, 1, n) \\
\text{return} \quad \text{median}
\end{align*}
\]

**FIND-MEDIAN(\( A, B, n, \text{low, high} \))**

\[
\begin{align*}
\text{if} \quad \text{low} > \text{high} &\quad \text{then} \quad \text{return} \quad \text{NOT-FOUND} \\
\text{else} \quad k &\leftarrow \lfloor (\text{low}+\text{high})/2 \rfloor \\
\text{if} \quad k = n \text{ and } A[n] \leq B[1] &\quad \text{then} \quad \text{return} \quad A[n] \\
\text{elseif} \quad k < n \text{ and } B[n-k] \leq A[k] \leq B[n-k+1] &\quad \text{then} \quad \text{return} \quad A[k] \\
\text{elseif} \quad A[k] > B[n-k+1] &\quad \text{then} \quad \text{return} \quad \text{FIND-MEDIAN}(A, B, n, \text{low, } k-1) \\
\text{else} &\quad \text{return} \quad \text{FIND-MEDIAN}(A, B, n, k+1, \text{high})
\end{align*}
\]