Homework 3 (70 points), Spring 2004

Q1: Exercise 2.3-3 (10 points)

Please draw the decision tree for quick sorting 3 input elements $< a, b, c >$.

- The tree is as Figure 1.

![Decision Tree Diagram](image)

Figure 1: The decision tree for quick sorting three elements

Q2: Exercise 8.1-3 (10 points)

Show that there is no comparison sort whose running time is linear for at least half of the $n!$ inputs of length $n$. What about a fraction of $1/n$ of the inputs of length $n$? What about a fraction of $1/2^n$?

- The running time of a specific input is linear if the corresponding leaf node is within a linear distance from the root. Considering a binary tree, the maximum number of nodes within a linear distance from the root is $2^{kn+1} - 1$.

- There is no comparison sort whose running time is linear for at least half of the $n!$ inputs because $n!/2$ is asymptotically bigger than $2^{kn+1} - 1$.

- There is no comparison sort whose running time is linear for at least a fraction of $1/n$ of the inputs because $n!/n$ is asymptotically bigger than $2^{kn+1} - 1$.

- There is no comparison sort whose running time is linear for at least a fraction of $1/2^n$ of the inputs because $n!/2^n$ is asymptotically bigger than $2^{kn+1} - 1$.

Q3: Exercise 8.2-1 (10 points)

Using Figure 8.2 as a model, illustrate the operation of Counting-Sort on the array $A = < 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 >$.

- The operation for counting sort is as Figure 2.
Figure 2: The operations for counting sort in Exercise 8.2-1

Q4: Exercise 8.2-4 (10 points)

Describe an algorithm that, given $n$ integers in the range 0 to $k$, preprocesses its input and answers any query about how many of the $n$ integers fall into a range $[a..b]$ in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

- Use line 1 to line 7 of the Counting-Sort algorithm (pp. 168) to preprocess the $n$ integers in the range 0 to $k$.
  After the preprocessing, we have an array $C[0..k]$ in which each $C[i]$ contains the number of integers less than or equal to $i$. Obviously, this preprocess takes $\Theta(n + k)$ time. Let Query(a, b) be an algorithm to return the number of the $n$ integers fall into a range $[a..b]$ in $O(1)$ time. The pseudocode for Query(a, b) is as follows.

```plaintext
Query(a, b) {
  if (b < a)
    then return error ‘invalid range’
  if (a > 0)
    then low-index = a - 1
    else low-index = -1
  if (b < k)
    then high-index = b
    else high-index = k
  if low-index = -1
    then return C[high-index]
    else return C[high-index] - C[low-index]
}
```
Q5: Exercise 8.3-1 (10 points)

Using the figure 8.3 as a model, illustrate the operation of Radix-Sort on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

- Shown on Figure 3.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COW</td>
<td>SEA</td>
<td>TAB</td>
<td>BAR</td>
</tr>
<tr>
<td>DOG</td>
<td>TEA</td>
<td>BAR</td>
<td>BIG</td>
</tr>
<tr>
<td>SEA</td>
<td>MOB</td>
<td>EAR</td>
<td>BOX</td>
</tr>
<tr>
<td>RUG</td>
<td>TAB</td>
<td>TAR</td>
<td>COW</td>
</tr>
<tr>
<td>ROW</td>
<td>DOG</td>
<td>SEA</td>
<td>DIG</td>
</tr>
<tr>
<td>MOB</td>
<td>RUG</td>
<td>TEA</td>
<td>DOG</td>
</tr>
<tr>
<td>BOX</td>
<td>DIG</td>
<td>DIG</td>
<td>EAR</td>
</tr>
<tr>
<td>TAB</td>
<td>BIG</td>
<td>BIG</td>
<td>FOX</td>
</tr>
<tr>
<td>BAR</td>
<td>BAR</td>
<td>MOB</td>
<td>MOB</td>
</tr>
<tr>
<td>EAR</td>
<td>EAR</td>
<td>DOG</td>
<td>NOW</td>
</tr>
<tr>
<td>TAR</td>
<td>TAR</td>
<td>COW</td>
<td>ROW</td>
</tr>
<tr>
<td>DIG</td>
<td>COW</td>
<td>ROW</td>
<td>RUG</td>
</tr>
<tr>
<td>BIG</td>
<td>ROW</td>
<td>NOW</td>
<td>SEA</td>
</tr>
<tr>
<td>TEA</td>
<td>NOW</td>
<td>BOX</td>
<td>TAB</td>
</tr>
<tr>
<td>NOW</td>
<td>BOX</td>
<td>FOX</td>
<td>TAR</td>
</tr>
<tr>
<td>FOX</td>
<td>FOX</td>
<td>RUG</td>
<td>TEA</td>
</tr>
</tbody>
</table>

Figure 3: The operations for radix sort in Exercise 8.3-1

Q6: Exercise 8.3-2 (10 points) and (extra credit 10 points)

Which of the following algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any sorting algorithm stable. How much additional time and space does you scheme entail?

- Part (a)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stable or not</td>
<td>insertion</td>
<td>merge</td>
<td>heap</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

- Part (b) A simple scheme makes any comparison sorting algorithm stable is described as follows.

A preprocess to the input makes each element a two field object (value, array index at the time of input). Then, for a stable sort, if two elements \((x_i, x_j)\) have the same value in the input, \((x_i, x_j)\) should appear before \((x_j, x_i)\) in the output if \(i < j\). We can always achieve that by defining two new comparison operators \(\gg\) and \(\ll\). The new operators works as follows.

\((x_i, x_j) \gg (y_i, y_j)\) either if \(x > y\) or if \(x = y\) and \(i > j\)

\((x_i, x_j) \ll (y_i, y_j)\) either if \(x < y\) or if \(x = y\) and \(i < j\)

To make all comparison sorts stable, we need to modify all sorting algorithms by replacing all \(>\) or \(\ge\) to \(\gg\), \(<\) or \(\le\) to \(\ll\) if they apply to elements.

Note: using the new operators, all elements are distinct.
Q7: Exercise 8.3-4 (10 points)

Show how to sort \( n \) integers in the range 0 to \( n^2 - 1 \) in \( O(n) \) time.

- Two digits are necessary to represent \( n^2 - 1 \) in the radix-\( n \). (that is, \( n-1 \) \( n-1 \))
  Each digit in the radix-\( n \) ranges from 0 to \( n-1 \). So \( k = O(n) \)
  The running time for Radix-Sort is \( \Theta(dn + kd) = \Theta(2n + 2O(n)) = \Theta(n) \)
  Therefore, \( n \) integers in the range 0 to \( n^2 - 1 \) can be sorted in \( O(n) \).