Homework #4: Sorting

CIS 121—Fall 2015
cis121-15fa-staff@googlegroups.com

Due: Thursday, October 1st, 2015 at 10:30 AM
(paper submission and online submission)

1 Stability—10 points

Prove or disprove: Insertion sort is stable. In your answer, define what it means for a sorting algorithm to be stable. Illustrate your definition by an example input along with two example outputs, one which has been sorted by a sorting algorithm that is stable and one which has been sorted by a sorting algorithm that is not stable.

2 Shell sort—20 points

1. Describe how Shell sort differs from Insertion sort.

2. Prove that if something is $h$ sorted after it is $k$ sorted for $h < k$ that it remains $k$ sorted.

3. Find a good sequence for Shell sort. Run the Shell.java program included in the homework tarball. Change the increment_sequence variable and evaluate the effect on the algorithm’s running time. Test the following sequences on an input sequence of 1 million random integers. Find a better sequence.

<table>
<thead>
<tr>
<th>increment sequence</th>
<th>running time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (This one will take a long time. You might want to run it overnight.)</td>
<td></td>
</tr>
<tr>
<td>524288, 262144, 131072, 65536, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1</td>
<td></td>
</tr>
<tr>
<td>524287, 262143, 131071, 65535, 32767, 16383, 8191, 4095, 2047, 1023, 511, 255, 127, 63, 31, 15, 7, 3, 1</td>
<td></td>
</tr>
<tr>
<td>797161, 265720, 88573, 29524, 9841, 3280, 1093, 364, 121, 40, 13, 4, 1</td>
<td></td>
</tr>
<tr>
<td>Your sequence here</td>
<td></td>
</tr>
</tbody>
</table>

4. Why the increment sequence containing only 1 take so long?

5. Why does the powers of 2 increment sequence work worse than the powers of 2 minus 1 increment sequence?

3 Invariants—10 points

What are the invariants of Insertion Sort? Give the invariants, and show how they are borne out in the code for Insertion Sort. Do the invariants still hold true for Shell Sort?
4 Quicksort trace—10 points

Give a trace of the behavior of quicksort. Fill out a table giving the values for lo, j, hi and the array using the specified input. Gray out the elements that are not being worked on during each partition. In your answer, say which implementation of quicksort you are using, either by reproducing the code or pseudocode for the algorithm or referencing the algorithm number in the textbook.

<table>
<thead>
<tr>
<th>lo</th>
<th>j</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

initial values

D A V I N C I C O D E

random shuffle

N I I E V C D C D O A

5 Heaviest segment—20 points

Given an input array \( A[1 .. n] \) of positive and negative real numbers, a segment of \( A \) is a portion of \( A \) of the form \( A[i .. j] \) for \( 1 \leq i \leq j \leq n \). The weight of a segment is the sum of the values of its elements. The heaviest segment problem determines the heaviest segment of a given input array \( A \).

1. Show that it is possible to determine the heaviest segment \( A[i .. j] \) of all segments having \( i = 1 \) in time \( O(n) \).
2. Give an algorithm for solving the heaviest segment problem in time \( O(n^2) \).
3. Give a divide and conquer algorithm in \( O(n \log n) \) running time for the problem.

6 Hat Trick—30 points

Consider the hat-check problem in which \( n \) customers give a hat to a hat-check person, called JJ, at a restaurant. JJ being new at his job does not keep the hats in an orderly fashion and when the customers come to collect their hats he does not know which hat belongs to which customer. It is also not possible for him to tell the relative sizes of heads of two customers or the relative sizes of the hats, which means that he cannot sort the hats nor can he sort the customers based on the sizes of their heads. The only test that JJ has is to have the customers try on a hat, and tell him whether the hat is too big, too small, or fits perfectly. All heads are of different sizes, so all hats are different in size. You may assume that each test takes unit time. In this problem, we want to design an efficient algorithm that JJ can use to return hats to the customers.

1. Consider the following algorithm to give the hats back to the customers. Let \( C \) denote the group of customers and \( H \) be the set of hats.

   1. if \( |H| = |C| = 1 \) then
   2.  give the hat to the customer
   3. else
   4.  Pick a hat \( h \) at random from \( H \)
   5.  for each customer \( c \) in \( C \) do
   6.    Customer \( c \) tries on hat \( h \)
   7.    if it fits then
   8.      Give hat \( h \) to customer \( c \)
   9.      \( C' = C - \{c\} \)
 10.      \( H' = H - \{h\} \)
 11.    break out of the for loop
 12. Hat-Check(\( C' \), \( H' \))
What is the expected running time of this algorithm? What is its worst case running time?

2. Design an algorithm with a better bound than the one in part (1). Be sure to explain your algorithm well, argue its correctness, and analyze its expected running time. Also, analyze its worst case running time.

3. Suppose the following: each time JJ gives a hat to a customer, the customer either says that the hat fit or it didn’t, without giving any information on whether it is too small or too big. Does your algorithm still work? Does the algorithm in part (1) still work?

7 Extra credit #1—10 points

Let $X[1 \ldots n]$ and $Y[1 \ldots n]$ be two arrays, each containing $n$ numbers already in sorted order. Give an $O(lg\ n)$-time algorithm to find the median of all $2n$ elements ($n$th smallest element) in the array $X \cdot Y$, where $X \cdot Y$ is the concatenation of the arrays $X$ and $Y$.

8 Extra credit #2—10 points

Max is a very studious 121 student. He makes an observation that some sorting algorithms do better on specific inputs. He decides that he can try to design an algorithm that takes advantage of this observation for “nearly” sorted arrays by modifying quicksort.

The algorithm goes something like this: Upon calling quicksort on a subarray with fewer than $k$ elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process.

However, Max doesn’t know how to analyze this algorithm, so he calls on his best friend Lewis. Lewis immediately exclaims, “This must run in $O(nk + n\ lg(n/k))$ expected time! Here let me show you…” but before he can write the proof, he runs off to help someone else with their algorithms homework. Can you help Max figure out how to prove this runtime?