CS 3343 (Spring 2014) Exam 2

April 15, 2014
9:30am - 10:50am (80 minutes)

Name: ______________________ ID: ______________________

- Don’t forget to put your name and ID on the cover page
- This exam is closed-book
- If you have a question, stay seated and raise your hand.
- Please try to write legibly – if I cannot read it, you may not get credit.
- Do not waste time – if you cannot solve a question immediately, skip it and return to it later.
- Try your best to answer each question. Partial credits will be given if you show that you have some ideas – but not according to the length of your answer.
- Be succinct.

<table>
<thead>
<tr>
<th></th>
<th>Sorting &amp; selection I</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sorting &amp; selection II</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Heaps</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Dynamic Programming I (LCS)</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic Programming II (RLP)</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Greedy algorithm</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Graph basics</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>(Extra credit) Algorithm Design</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>110</td>
</tr>
</tbody>
</table>
1. (18 points) Sorting and Selection (Part I)
   Indicate whether the following statement is true or false. Briefly justify your answers.
   a. Merge sort is more efficient than heap sort asymptotically, because the former is a stable sorting algorithm while the latter is not.

   b. The running time of counting sort depends on the values of the input and is not always $\Theta(n)$.

   c. Quick sort runs in linear time for an already sorted array.

   d. In theory, quick sort can be implemented to run in $\Theta(n \log n)$ time even in the worst case, for example, by using the Linear Time Select algorithm to always find median as pivot.

   e. No sorting algorithm can be designed to satisfy the following three requirements simultaneously: (1) has $\Theta(n \log n)$ running time, (2) is stable, and (3) is in-place.

   f. Quick sort and randomized quick sort have the same asymptotic worst-case running time.

   g. Finding the median from a sorted array takes the same amount of time as from an unsorted array.

   h. The advantage of a heap is that it enables you to get all elements in an array into sorted order in $\Theta(n)$ time.

   i. In the worst case, randomized select has the same asymptotic running time as randomized quick sort.
2. (10 points) Sorting and Selection (Part II)

Suppose that you have an array of \( n \) numbers, and you would like to get the \( k \) smallest numbers in sorted order. You are considering the following three options:

(i) Sort the numbers into non-decreasing order using merge sort and then return the first \( k \) numbers.

(ii) Build a min-heap and then call Extract-Min \( k \) times.

(iii) Use the randomized select algorithm to find the \( k \)-th smallest number, partition around that number, and sort the \( k \) smallest numbers.

a. Analyze the running time of the above three methods in terms of both \( n \) and \( k \),

b. Which method(s) would you prefer and why, when \( k = \Theta(1) \), i.e., \( k \) is a relatively small number independent of \( n \) (e.g., \( k = 10 \))?

c. Which method(s) would you prefer and why, when \( k = \Theta(n) \), i.e., \( k \) is proportional to \( n \) (e.g., \( k = n/4 \)).

3. (16 points) Heaps

a. Does the array \([10 \ 9 \ 7 \ 4 \ 8 \ 3 \ 6 \ 3 \ 2 \ 7] \) represent a max heap? Why or why not?
b. Consider the following two procedures for building a max heap. Both algorithms take an unsorted array A as an input and make A a heap.

<table>
<thead>
<tr>
<th>BuildHeap(A) {</th>
<th>BuildHeap'(A) {</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap_size(A) = length(A);</td>
<td>heap_size(A) = 1;</td>
</tr>
<tr>
<td>for (i = floor(length[A]/2) downto 1)</td>
<td>for (i = 2 to length[A])</td>
</tr>
<tr>
<td>Heapify(A, i);</td>
<td>HeapInsert(A, A[i]);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

(I) Illustrate how the procedure BuildHeap works on an array [1 5 6 4 2 7 3]. Show necessary intermediate steps for full credit (e.g., show the content of the tree after each Heapify).

(II) Illustrate how the procedure BuildHeap’ works using the same example. Show necessary intermediate steps for full credit (e.g., show the content of the heap after each HeapInsert).

(III) Do the two procedures BuildHeap’ and BuildHeap have the same time complexity asymptotically? Briefly justify your answer.
4. (18 points) Longest common subsequence (LCS).

a. (15 points) Use dynamic programming to find LCS between two strings ACDBDCAB and ABDBCDAC. If there are multiple longest common subsequences, report all of them.

b. (3 points) The running time of LCS is \( \Theta(mn) \), where \( m \) and \( n \) are the lengths of the two strings, respectively. This time complexity is valid only if you are interested in finding a longest common subsequence, even if multiple exist. Argue (or use an intuitive example to show) that if you are interested in returning ALL longest common subsequences, the running time may be much longer.
5. (21 points) Restaurant location problem.

a. (15 points) Solve the following optimal restaurant location problem using dynamic programming. The distance constraint is that two selected locations cannot be within 10 miles. Note that \( d_i \) is the distance between location \( i \) and location 1.

<table>
<thead>
<tr>
<th>Location ( i )</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Distance ( d_i )</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>12</td>
<td>15</td>
<td>21</td>
<td>27</td>
<td>33</td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>Profit ( p_i )</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

\begin{array}{|c|c|c|c|c|c|c|c|c|c|}
\hline
select = no & & & & & & & & & & \\
\hline
select = yes & & & & & & & & & & \\
\hline
S(i)         & & & & & & & & & & \\
\hline
\end{array}

Optimal value:

Locations (indices) selected in an optimal solution:

b. (6 points) Now consider a special case of the restaurant location problem where all the candidate locations are evenly spaced. (For example, assume the distance between any two consecutive locations is exactly 5 miles.) Design a dynamic programming strategy to solve the problem in linear time. Write down your recurrence, and use it to solve the above problem after changing the distances so that \( d_i = 0, 5, 10, 15, 20, \ldots \).
6. (12 points) Greedy algorithm.

a. (9 points) Use greedy algorithm to solve the fractional knapsack problem. The knapsack has a weight limit of 10 LBs.

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Weight (LB)</th>
<th>Value ($)</th>
<th>Value / Weight ($/LB)</th>
<th>Weight (LB) taken</th>
<th>Value ($) taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>35</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>18</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>12</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>9</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>10</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

b. (3 points) If the objective of the problem is changed to filling the knapsack with the least (instead of most) total value, can greedy algorithm still be applied? Why or why not?

7. (5 points) Graph basics.

Draw a graph that can be represented by the adjacency matrix below.

```
0 1 0 0 0 0 0 0
1 0 0 1 0 1 0 0
0 0 0 1 1 0 0 0
0 1 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 1 1 1 0 0
```
8. (Extra Credit - 10 points) Longest common subsequence among three sequences.
The problem is to find a longest common subsequence among three strings, X, Y, and Z.

a. One simple idea is to use dynamic programming to first find LCS between two sequences, say, X, and Y, and then find the LCS between LCS(X, Y) and Z. Argue (or give an example to show) why this algorithm may not always find the optimal solution.

b. Describe an efficient dynamic programming algorithm that can solve the problem, and analyze its running time. (Assume that all three sequences have the same length n.)
Blank page