CS 3343.002 (Fall 2016) Exam 2

Nov 17, 2016
2:30pm - 3:50pm (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; order statistics</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>2)</td>
<td>Sorting &amp; order statistics II</td>
<td>10</td>
</tr>
<tr>
<td>3)</td>
<td>Analysis of randomized algorithm</td>
<td>10</td>
</tr>
<tr>
<td>4)</td>
<td>Heap</td>
<td>15</td>
</tr>
<tr>
<td>5)</td>
<td>Dynamic programming I (LCS)</td>
<td>15</td>
</tr>
<tr>
<td>6)</td>
<td>Dynamic programming II (RLP)</td>
<td>20</td>
</tr>
<tr>
<td>7)</td>
<td>Greedy algorithm</td>
<td>10</td>
</tr>
<tr>
<td>8)</td>
<td>(Extra credit) Algorithm design</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>130</td>
</tr>
</tbody>
</table>
1. (20 points) Sorting and Order Statistics (I)

Indicate whether the following statement is true or false. Justification is NOT required.

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

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

c. Counting sort is more efficient than other sorting algorithms, because its running time is $\Theta(n)$ while the others are at least $\Theta(n \log n)$.

d. Counting sort is a stable and in-place sorting algorithm.

e. Merge sort is more efficient than heap sort asymptotically, because the former is a stable sorting algorithm while the latter is not.

f. 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.

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

h. In the worst case, Randomized SELECT has the same running time as randomized quick sort.

i. To find the largest element in an array, the most efficient approach is to build a max heap and do ExtractMax.

j. 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.
2. (10 points) Sorting and Order Statistics (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. (6 points) Analyze the running time of the above three methods in terms of both \( n \) and \( k \).

b. (2 points) 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. (2 points) 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. (10 points) Analysis of randomized algorithm
Consider the following randomized algorithm which takes an integer \( n \) as input.

```
RandAlg(n)
    if (n <= 1) return;
    p = 0.5;
    r = rand(); // r is a uniform random number between 0 and 1
    if (r <= p) // with probability p
        RandAlg(n/2);
    else { // with probability 1-p
        RandAlg(n/2);
        RandAlg(n/2);
    }
```

a. What is the worst case running time of RandAlg?

b. What is the best case running time of RandAlg?

c. What is the expecting running time of RandAlg?

d. If line 3 is changed to “\( p = 0.9 \)”, what will be the expected running time of RandAlg?

e. What is the expected running time of RandAlg as a function of \( p \)? When \( p \) increases, does the running time increase or decrease?
4. (15 points) Heap

a. (2 points) Does the array [10 8 9 4 5 6 7 1 2 3] represent a max heap? Why or why not?

b. (2 points) Given an array representing a max heap, can we reverse the order of the elements in the array to obtain a min heap? Why or why not?

c. Compare the following two procedures for building a max heap (BuildHeap1 is what we learned in class). Both algorithms take an unsorted array A as an input and make A a heap.

<table>
<thead>
<tr>
<th>BuildHeap1(A)</th>
<th>BuildHeap2(A)</th>
</tr>
</thead>
</table>
| \[
\text{heapSize}(A) = \text{length}(A); \\
\text{for } (i = \text{floor}(\text{length}[A]/2) \text{ downto } 1) \\
\quad \text{Heapify}(A, i);
\] | \[
\text{heapSize}(A) = 1; \\
\text{for } (i = 2 \text{ to } \text{length}[A]) \\
\quad \text{HeapInsert}(A, A[i]);
\] |

(I) (3 points) Illustrate how the procedure BuildHeap1 works on an array [1 5 6 4 2 7 3]. Show the content of the array/heap after each Heapify.

(II) (4 points) Illustrate how the procedure BuildHeap2 works using the same example. Show content of the array/heap after each HeapInsert.
(III) (2 points) What is the time complexity of BuildHeap2 and how does it compare to BuildHeap1?

(IV) (1 points) If we change the third line of BuildHeap1 to “for (i = 1 to length[A])”, would the algorithm still work? If yes, what is the time complexity of the modified algorithm?

(V) (1 points) If we change the fourth line of BuildHeap2 to the following two lines:

```c
heapSize(A)++; 
heapify(A, i);
```

would the algorithm still work? If yes, what is the time complexity of the modified algorithm?

5. (15 points) Longest common subsequence (LCS).

a. (12 points) Use dynamic programming to find LCS between two strings BAABAB and ABBAAB. In your trace-back, show only one path (if there are multiple), and the LCS corresponding to that path.

```
<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>A</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</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>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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 if you are only interested in returning a longest common subsequence, even if multiple exist. Argue (or use an intuitive example to show) that if you want to report ALL longest common subsequences, the running time may be much longer.
6. (20 points) Restaurant location problem.

a. (12 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>
<tr>
<td>select = yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>select = no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S(i)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimal value:

Locations (indices) selected in an optimal solution:

b. (2 points) Suppose that later you found that due to a small problem in the estimation process, the estimated revenue may not be very accurate. Specifically, you had underestimated the revenue of each location by 10%. Do you need to rerun your algorithm to get the optimal solution? Why or why not?

c. (2 points) Similar to (c), but you found that you had underestimated the revenue at each location by a fixed number (say, 1), do you need to rerun your algorithm to get the optimal solution? Why or why not?

d. (4 points) Consider a modification to the policy about the distance between selected locations. The new policy says that you CANNOT have three consecutive locations within 10 miles from its neighbors, i.e., for any three selected locations $i < j < k$, you CANNOT have
both \(d_k - d_j < 10\) and \(d_j - d_i < 10\). (For example, if loc 1 and loc 2 is less than 10 miles away from each other and you selected both, then the next one you selected must be at least 10 miles from loc 2.) Provide a dynamic programming algorithm to solve the problem.

Hint: at each location \(i\), consider three possible cases: (1) location \(i\) is not selected, (2) location \(i\) is selected but no other locations within 10 mile has been selected, and (3) location \(i\) is selected, and another location within 10 miles may have also been selected.

Let \(S(i)\) be the optimal value selected up to location \(i\), \(P(i, k)\) be the optimal value selected up to location \(i\) in case \(k\) above, and \(j_i < i\) be the largest index of the location that is at least 10 miles away from location \(i\).

Define \(S(i), P(i, 1), P(i, 2)\) and \(P(i, 3)\) recursively.

7. (10 points) Greedy algorithm.

a. (7 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 ($)</th>
<th>$/LB</th>
<th>Weight (LB) taken</th>
<th>Value ($) taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></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?
8. (30 points) Extra credit: Algorithm design.

a. (10 points) **Sorting.** Given $d$ sequences of elements where each sequence is already sorted and the total number of elements in the $d$ sequences is $n$, design an $\Theta(n \log d)$ algorithm to merge all the sequences into one sorted sequence. e.g., merging “1 3”, “2 5”, and “4 6” gives us “1 2 3 4 5 6”. (For full credit, show both the algorithm and the analysis of its running time.)

b. (10 points) **Order Statistics.** Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing $n$ numbers already in sorted order. Describe an $\Theta(\log n)$-time algorithm to find the median of the $2n$ elements in arrays $X$ and $Y$.

c. (10 points) **Dynamic Programming.** A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Design an efficient algorithm that takes a sequence $x[1 \ldots n]$ and returns the **longest palindromic subsequence**. What is the running time of your algorithm?

(For instance, the sequence ACGTGTCAAAATCG has many palindromic subsequences, including ACGCA and CAAC. On the other hand, the subsequence ACG is not palindromic.)
Scratch paper
Scratch paper
Scratch paper