CS800,  Solutions to the  Practice Midterm

Name: ________________________

- This exam is worth 40 points. It consists of five problems, each worth 10 points. The sum of the four highest scored problems defines the final grade.

- If you want to ask a question, write it on the provided sheet of paper and raise your hand. I will collect it, write down the answer and return it back to you.

- If you need any more scrap paper, raise your hand. Use only the scrap paper provided.

- Please turn off your cell-phones and other electronic devices.

- When time is called, stop writing immediately and remain seated until all exams have been collected.

- If you finish during the last 10 minutes of the exam, please remain seated until the end of the exam.
Problem 1

Rank the following functions by order of growth; that is, find an arrangement \( g_1(n), g_2(n), \ldots, g_{10}(n) \) of functions satisfying \( g_i(n) = \Omega(g_{i+1}(n)) \) for every \( i \in \{1, \ldots, 9\} \). Partition your list into equivalence classes such that \( f(n) \) and \( g(n) \) are in the same class if and only if \( f(n) = \Theta(g(n)) \). You do not have to prove your answers.

\[
\begin{array}{cccccc}
\text{n log}_3 n & n^{5.3} + 2n^7 & n \log(n^2) & 3^n & 5\sqrt{n} \\
\frac{1}{10000}n^7 & 1 & n^{-3} & 2^\log n & 1 + 4n^{-3}
\end{array}
\]

Recall that in this class we use \( \log n \) to denote the logarithm base 2.

\[
\begin{array}{cccccc}
3^n, & 5^n, & n^{5.3} + 2n^7, & \frac{1}{10000}n^7, & 2^\log n \\
The same equivalence class
\end{array}
\]

\[
\begin{array}{cccccc}
\log_3 n, & n \log(n^2), & 1 + 4n^{-3}, & 1, & n^{-3} \\
The same equiv. class
\end{array}
\]

Overall: 6 equivalence classes
Problem 2

Given are three sorted arrays \( A, B, \) and \( C, \) with a total of \( n \) elements. Give an \( O(n) \) algorithm which determines if there is an element which occurs in all three arrays.

Include a verbal description of your algorithm and reason its running time. You may use any algorithms from class as subroutines.

1. \( a = A.\text{length}; \quad b = B.\text{length}; \quad c = C.\text{length}; \)
2. \( A[a+1] = \infty; \quad B[b+1] = \infty; \quad C[c+1] = \infty; \)
3. \( i=1; \quad j=1; \quad k=1; \)
4. \( \text{while} \ (i+j+k < n+3) \{ \)
5. \( \quad \text{if} \ (A[i] = B[j] = C[k]) \quad \text{then return} \ TRUE \)
6. \( \quad \text{if} \ (A[i] \leq B[j] \text{ and } A[i] \leq C[k]) \quad \text{then} \ i++ \)
7. \( \quad \text{else if} \ (B[j] \leq A[i] \text{ and } B[j] \leq C[k]) \quad \text{then} \ j++ \)
8. \( \quad \text{else} \ k++ \)
9. \( \quad \text{return} \ FALSE \)

**Explanation:**

Idea is similar to the MERGE algorithm. We will keep a pointer in every array, the pointer points to the currently examined elements in that array. If all three elements specified by the pointers are equal, we have found an element that occurs in every array. Otherwise, one of the three elements is the smallest and it cannot be an element occurring in every array - so we move the corresponding pointer to the next element in that array.

**Running time:**

The while loop runs for at most \( n \) steps and inside the loop we make only \( O(1) \) steps. Thus, the overall running time is \( O(n) \).
Problem 3

We are given a sequence of $n$ positive numbers $a_1, \ldots, a_n$. Give a dynamic programming algorithm which finds the length of the longest increasing subsequence of $a_1, \ldots, a_n$ with the property that two consecutive elements have to differ by at least 2. (For example on input 1, 101, 2, 3, 100, 4, 5 your algorithm should output 3, the length of the subsequence 1, 3, 5.) The running time of your algorithm should be $O(n^2)$.

Make sure to describe the “heart of the algorithm” which is worth 5 points. If you do not include the heart of the algorithm, your solution will get no credit.

Heart of the algo:

$$S[k] = \text{the length of the longest increasing subsequence of } a_1, \ldots, a_k \text{ ending with } a_k \text{ and satisfying the condition that all consecutive elements differ by } \geq 2$$

$$S[k] = \max \{ 1, \max_{i < k, a_i \leq a_k - 2} S[i] + 1 \}$$

Algorithm:

for $k = 1$ to $n$ do

$S[k] = 1$

for $i = 1$ to $k-1$ do

if $a[i] \leq a[k] - 2$ and $S[i] + 1 > S[k]$ then

$S[k] = S[i] + 1$

return $\max_{k=1}^{n} S[k]$
Problem 4

Professor Wise observed that his files consist of only letters A, B, C, D, E, F with the following frequencies:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>5%</td>
<td>5%</td>
<td>15%</td>
<td>20%</td>
<td>25%</td>
<td>30%</td>
</tr>
</tbody>
</table>

Professor Wise wanted to save some space on his drives so he decided to use a prefix-free (binary) code to encode the above letters. After a significant thinking time he came up with the following code:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>001</td>
<td>000</td>
<td>010</td>
<td>011</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

However, Professor Wise is still unsure if his code is optimal. Help him by answering these questions:

- Is Professor Wise’s code prefix-free? **YES.**

- What is the expected length of a codeword in Professor Wise’s code? Give a numerical answer or a fraction.

  \[3 \cdot \frac{5}{100} + 3 \cdot \frac{5}{100} + 3 \cdot \frac{15}{100} + 3 \cdot \frac{20}{100} + 2 \cdot \frac{25}{100} + 2 \cdot \frac{30}{100} = \frac{49}{20}\]

- Using an algorithm from class compute an optimal prefix-free code for the above input. Describe the algorithm (2-3 sentences, no pseudocode) and state its name.

  **OPTIMAL CODE:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>01</td>
</tr>
</tbody>
</table>

- What is the optimal expected length of a codeword?

  \[4 \cdot \frac{5}{100} + 4 \cdot \frac{5}{100} + 3 \cdot \frac{15}{100} + 2 \cdot \frac{20}{100} + 2 \cdot \frac{25}{100} + 2 \cdot \frac{30}{100} = \frac{47}{20}\]
Problem 5

Given is an array of integers $A = [a_1, a_2, \ldots, a_n]$. Consider the following two algorithms for finding the maximum value in $A$.

a) $MAX(A)$

1. If $n = 1$ then return $a_1$.
2. Let $k = \lceil (1 + n)/2 \rceil$.
3. Create arrays $B[1..k] = [a_1, a_2, \ldots, a_k]$ and $C[1..(n - k)] = [a_{k+1}, a_{k+2}, \ldots, a_n]$.
4. Let $m_B = MAX(B)$ and $m_C = MAX(C)$.
5. Return $\max\{m_B, m_C\}$.

Let $T(n)$ be the running time of $MAX$ on an array with $n$ elements. Give the recurrence for $T(n)$:

$$T(n) = \begin{cases} 
0(1) & \text{if } n = 1 \\
T(\lceil \frac{n}{2} \rceil) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & \text{if } n > 1 
\end{cases}$$

Give the running time $T(n)$ evaluates to and reason why:

$$T(n) = \Theta(n \log n)$$

We can use the Master Theorem:

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

(if we ignore rounding)

Thus,

$a = 2$, $b = 2$ and $f(n) = \Theta(n)$.

Then, $f(n) = \Theta(n \log^{a/b} n) = \Theta(n)$ and $T(n) = \Theta(n \log^{a/b} \log n) = \Theta(n \log n)$

(Incidentally, this is the same recurrence as for MERGESORT.)
b) $MAX_2(A, \ell, r)$

1. If $\ell = r$ then return $a_\ell$.
2. Let $k = \lfloor (\ell + r)/2 \rfloor$.
3. Let $m_\ell = MAX_2(A, \ell, k)$ and $m_r = MAX_2(A, k + 1, r)$.
4. Return $\max\{m_\ell, m_r\}$.

In this algorithm, the content of $A$ does not need to be copied for the recursive calls. To compute the maximum of $A$ we simply call $MAX_2(A, 1, n)$.

Let $T_2(s)$ be the running time of $MAX_2$ on $s$ elements, i.e. $s = r - \ell + 1$. Give the recurrence for $T_2(s)$:

$$T_2(s) = \begin{cases} 
  O(1) & \text{if } s = 1 \\
  T_2(\lfloor \frac{s}{2} \rfloor) + T_2(\lceil \frac{s}{2} \rceil) + O(1) & \text{if } s > 1
\end{cases}$$

Give the running time $T_2(s)$ evaluates to and reason why:

$$T_2(s) = \Theta(s)$$

We can use the Master Theorem as before: $a = 2, b = 2, f(s) = O(1)$.

Thus $f(s) = O(s \log_s 2^{a-1})$ for e.g. $\varepsilon = 1$.

Then, $T_2(s) = \Theta(s \log_s 2^2) = \Theta(s)$.

Note: Both recurrences can be also analyzed, e.g. using a proof by induction, or using the substitution method – both were discussed in class (substitution for HERGESORT, induction for SELECT).