College of Computer & Information Science
Northeastern University
Spring 2019

CS5800
Algorithms
March 8, 2019

Sample Solution to Midterm 1

1. (5 points) Stable Matching

True or False?: There exists an instance of the stable matching problem with 3 men and 3 women, which has a stable matching in which every man is engaged to his least preferred woman and every woman is engaged to her least preferred man.
If your answer is True, then you need to give a specific example. If your answer is False, then you need to prove that no such instance exists.

Answer:
False. Suppose, for the sake of contradiction, there exists an instance of the stable matching problem with 3 men and 3 women, which has a stable matching in which every man is engaged to his least preferred woman and every woman is engaged to her least preferred man. Let $M = (m_1, w_1), (m_2, w_2), (m_3, w_3)$ be one such stable matching. Then, since m_1 prefers w_2 more than w_1, and w_2 prefers m_1 more than m_2, M has an instability, leading to a contradiction.

2. (3 + 3 = 6 points) Asymptotic Notation

(a) Sort the following functions in the order of their asymptotic growth:

$$n^{1.5},\ n\log n,\ \frac{n^2}{\log n}$$

Answer:
The correct ordering will be:

$$n\log n < n^{1.5} < \frac{n^2}{\log n}$$

i.e., $n\log n = O(n^{1.5})$ and $n^{1.5} = O(\frac{n^2}{\log n})$.

We will prove each in turn.

$$\lim_{n\to\infty}\frac{n\ \log n}{n^{1.5}} = \lim_{n\to\infty}\frac{\log n}{\sqrt{n}}$$

Using L'Hospital's rule, we have:

$$\lim_{n\to\infty}\frac{\log n}{\sqrt{n}} = \lim_{n\to\infty}\frac{\frac{1}{n}}{0.5n^{-0.5}} = \lim_{n\to\infty}\frac{2}{\sqrt{n}} = 0 \qquad (1)$$

This proves $n\log n = O(n^{1.5})$.

$$\lim_{n\to\infty} \frac{n^{1.5}}{\frac{n^2}{\log n}} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0 \ \ (\mathit{from}\ (1)\ \mathit{above})$$

This proves $n^{1.5} = O(\frac{n^2}{\log n})$. QED.

(b) Let $f(n)$ and $g(n)$ be asymptotically positive and monotonically increasing functions. Decide whether the following statement is true, and give a proof or a counterexample.

If $f(n) = \Omega(g(n))$, then $\sqrt[3]{f}(n) = \Omega(\sqrt[3]{g}(n))$.

Answer:
True. Since $f(n) = \Omega(g(n))$, there exist $c, n_0 > 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$. We thus have:

$$\sqrt[3]{f}(n) \geq \sqrt[3]{c}\ \sqrt[3]{g}(n)$$

for all $n \geq n_0$. We thus have $\sqrt[3]{f}(n) = \Omega(\sqrt[3]{g}(n))$.

3. (5 points) Recurrence Relations

Derive an asymptotically tight bound for the following recurrence. Assume that $T(n)$ is $\Theta(1)$ for $n \leq 4$.

$$T(n) = T(n/4) + T(3n/4) + n^2.$$

Answer: Use recursion tree approach. First level would contribute n^2. Second level $13n^2/16$. Third level $(13/16)^2n^2$. So we have

$$T(n) \leq n^2 + (13/16)n^2 + (13/16)^2n^2 + \cdots \leq n^2/(1 - 13/16) = O(n^2).$$

T(n) is clearly $\Omega(n^2)$ since it is at least n^2. So $T(n) = \Theta(n^2)$.

Problem 4. (6 points) Finding a specific index in a sorted array

You are given a *sorted* array of *distinct* integers $A[1 \ldots n]$. These integers could be negative or nonnegative.

Give an $O(\log n)$ time algorithm to find out whether there is an index i such that $A[i] = i$. For partial credit, you may give an algorithm with a worse running time. State the worst-case running time of your algorithm (no analysis needed).

Answer:
Variant of binary search. Look at middle element at index, say m. If middle element is greater than m, then search in smaller half; if middle element is smaller than m, then search in larger half. If middle element is m, then return middle index.

Problem 5. (8 points) Finding the Local Maximum

Given a set of $n \geq 2$ distinct numbers in an array $A[0 \ldots n-1]$, the element $A[i]$ is called a local maximum if one of the following holds:

- $i = 0$ and $A[0] > A[1]$, or
- $i = n - 1$ and $A[n-1] > A[n-2]$, or
- $0 < i < n - 1$ and $A[i-1] < A[i]$ and $A[i] > A[i+1]$

Note that while the maximum element of A is always a local maximum of A, the converse may not be true. For example, given the array $A = [12, 15, 8, 22, 36, 19, 17, 54, 63, 7, 42], A[1] = 15, A[4] = 36, A[8] = 63$, and $A[10] = 42$ are all local maxima.

Give an algorithm to determine a local maximum of an array A of numbers of length n. If A has multiple local maxima, your algorithm may return any one of them. Derive a tight Θ-bound on your algorithm's worst-case running time, which is defined as the number of elements of A accessed by your algorithm.

Your grade will be determined on the basis of the correctness of your algorithm and its efficiency, given by its worst-case running time. Partial credit may be given for non-optimal algorithms provided they are correct and well explained.

Answer:
Here is an algorithm to find a local maximum of A.

Algorithm 1: Find the Local Maximum of a Set of Distinct Numbers

```
FindLocalMaximum($A, L, U$)
Comment: Find a local maximum between A[L] and A[U]
if $L = U$ then
    return $A[L]$ Comment: Return the only element in range
end
if $A[L] > A[L+1]$ then
    return $A[L]$ Comment: Return leftmost element as Local Maximum
end
if $A[U] > A[U-1]$ then
    return $A[U]$ Comment: Return rightmost element as Local Maximum
end
$M \leftarrow \lfloor \frac{L+U}{2} \rfloor$ Comment: Find the midpoint
if $A[M] > A[M-1]$ and $A[M] > A[M+1]$ then
    return $A[M]$ Comment: Return the middle element $M$ if it is a local maximum
else if $A[M-1] > A[M]$ then
    FindLocalMaximum($A, L, M-1$)
else
    FindLocalMaximum($A, M+1, U$)
end
```

Invoke FindLocalMaximum$(A, 0, n-1)$ to solve problem as stated.

Strategy: Use recursion reducing the size of the array being searched in half on each recursive call. The base cases:

(i) A single element in the array, in which case the only element is the local maximum

(ii) The leftmost element is larger than the adjacent element to the right, in which case the leftmost element is a local maximum.

(iii) The rightmost element is larger than the adjacent element to the left, in which case the rightmost element is a local maximum.

Recursive step: If none of the base cases apply, there are at least 3 elements in the array and the local maxima are between left and right ends of the array. Find the middle element and compare it with the adjacent elements on the left and right. If the middle element is larger than elements on either side, then it is one of the local maxima and is returned. Otherwise, if the element to the left is larger, there must be a local minimum on the left half of the array. Else, if the element to the right is larger, there must be a local minimum in the right half of the array.

The time complexity of the algorithm is given by

$$T(n) = T(n/2) + \Theta(1)$$

and can be solved using Master Theorem as $T(n) = \Theta(\log\ n)$.