College of Computer & Information Science — CS5800
Northeastern University — Algorithms
Fall 2018 — October 11, 2018

Solution Sketch for Practice Problems for Midterm 1

1. Properties of a stable matching

For each of the following two items, justify your answer.

(a) 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 most 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: True. Consider an instance with the following preference lists.

man	first	second	third
m_1	w_3	w_2	w_1
m_2	w_1	w_3	w_2
m_3	w_2	w_1	w_3

woman	first	second	third
w_1	m_1	m_2	m_3
w_2	m_2	m_3	m_1
w_3	m_3	m_1	m_2

Then, the matching $\{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$ is stable since every woman is matched to her most preferred man, and hence does not to swap. The matching satisfies the required condition that every man is engaged to his least preferred woman and every woman is engaged to her most preferred man.

(b) 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. Asymptotic notation

- Prove or disprove the following statement:

$$f(n) = \Theta((f(\sqrt{n}))^2)$$

for all asymptotically positive, monotonically increasing functions $f(n)$.

Answer: This is false. Consider $f(n) = \log n$. We obtain

$$(f(\sqrt{n}))^2 = (\log(\sqrt{n}))^2 = \frac{\log^2 n}{4}.$$

We observe that $\log n$ is not $\Theta\left(\frac{\log^2 n}{4}\right)$ since

$$\lim_{n\to\infty} \frac{\log n}{(\log^2 n)/4} = 0.$$

- Compare the following pair of functions in terms of asymptotic growth rate.

$$f(n) = n2^n; \;\; g(n) = 3^n.$$

Answer: We have $f(n) = O(g(n))$ since

$$\lim_{n\to\infty} \frac{n2^n}{3^n} = \lim_{n\to\infty} \frac{n}{1.5^n} = \lim_{n\to\infty} \frac{1}{1.5^n \ln(1.5)} = 0.$$

3. Recurrence

Derive an asymptotically tight bound for the following recurrences. Assume that $T(n)$ is $\Theta(1)$ for $n \leq 4$. You may ignore floors and ceilings in your calculations.

(a) $T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor 3n/4 \rfloor) + n^2$.

Answer: One can use the recursion tree approach to obtain a geometrically decreasing sequence, with n^2 at the top level. This leads to a $\Theta(n^2)$ bound.

One can prove this formally by the substitution method. The lower bound is immediate. The upper bound can be established by induction.

(b) $T(n) = 16T(\lfloor n/4 \rfloor) + n \log n$.

Answer: We apply the Master Theorem. We have $n^{\log_b a} = n^2$, and $f(n) = n \log n$. We see that $n \log n = O(n^{2-\epsilon})$ for any $eps < 1$. For instance, take $\varepsilon = 0.5$. Then, $(n \log n)/n^{1.5} = \log(n)/n^{0.5}$. We can take the limit as $n \to \infty$, and applying L'Hopital's rule once, we obtain that the limit of the ratio is 0. So $n \log n = O(n^{2-0.5})$. Applying the relevant case of Master Theorem, we obtain $T(n) = \Theta(n^2)$.

One can also establish this using the recursion tree method. We obtain a geometrically increasing sequence, with $n^{\log_4(16)} = n^2$ at the last level dominating the contributions of other levels.

4. Sorting a zig-zag array

We say that an array $A[0 \ldots n-1]$ of distinct integers is *zig-zag*, if there exists an index i, $0 \leq i < n$ such that if we append the sub-array $A[0 \ldots i-1]$ after sub-array $A[i \ldots n-1]$, we obtain an array sorted in increasing order. Put it another way, A is zig-zag if the array $B[0 \ldots n-1]$ given by $B[j] = A[(j+i) \bmod n]$ for $0 \leq j < n-1$ is a sorted array (in increasing order).

For example, the arrays $[3, 5, 9, -1, 0, 2]$, $[1, 2, 3, 4, 5, 6]$, and $[9, 1, 3, 5, 6, 7]$ are all zig-zag arrays while $[-1, 8, 5, 4, 9, 0]$ is not a zig-zag array.

Give an algorithm that takes as input a zig-zag array $A[0 \ldots n-1]$ and determines the smallest element of A, while accessing at most $O(\log n)$ elements of A. For partial credit, you may give an algorithm that has a worse bound on the number of elements of A it accesses.

Answer: We will use binary search and find the index with the largest element and return the one following it (circularly, if needed).

```
1. $\ell = 0, r = n-1$.
2. while $r - \ell \geq 0$:
    (a) $m = \lfloor (\ell + r)/2 \rfloor$.
    (b) if $A[m] > A[\ell]$, then $\ell = m$, else $r = m - 1$.
3. return $A[(\ell + 1) \bmod n]$.
```

Worst-case running time is $O(\log n)$.

5. Mode of an array

A *mode* of an array $A[1..n]$ is an element of A that occurs most number of times in A. Thus, the mode of an array $A = [7, 4, 12, 4, 1, 1, 4]$ is 4, which occurs three times.

Give an algorithm to compute the mode of a given array $A[1..n]$ of n integers. Analyze the worst-case running time of your algorithm.

Your grade for this question 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: Sort the n numbers in nondecreasing order. Go through the n numbers in sorted order, keeping a count of the number of occurences of the current element while also maintaining the element that has occured the maximum number of times thus far (also storing its count). At the end of the process, return the element that has occured the maximum number of times.

Sorting takes $\Theta(n \log n)$ time using mergesort. The linear scan through the sorted list takes $\Theta(n)$ time. So the total time is $\Theta(n \log n)$.

An alternative *randomized* algorithm uses hash tables. We do a linear scan through the elements, incrementing the count for the current element in its slot stored in the hash table. We can also maintain the maximum count (with the element that has this count) thus far through this scan, updating the maximum when we find an element that has a higher count than the current maximum. Since inserting into and searching a hash table takes expected $\Theta(1)$ time, the expected total time is $\Theta(n)$.