College of Computer & Information Science
Northeastern University
Spring 2019

CS5800
Algorithms
21 February 2019

Sample Solution to Midterm 1

1. (5 points) Stable Matching

In a summer internship program organized by tech companies, there are n tech companies and each of them has one open internship position for summer of 2019. There are n graduate students who are qualified to participate in this program. Each of the tech companies ranks all of the student applicants by preference, and each of the students ranks all of the tech companies according to their preference.

Among the n tech companies, k are located in the Bay area, where $k < n$. Coincidentally, precisely k of the student applicants also live in the Bay area. Companies in the Bay area rank the k students who live in the area higher than students who live in other locations. Similarly, the k students who live in the Bay area prefer to do their internship in the same area where they live, and rank the companies from the Bay area higher than other companies.

Prove that in every stable matching for this instance, all of the k students who lives in the Bay area are assigned to a company located in the same area.

Answer:
We prove that for this instance, in every stable matching, all of the k students from the Bay area are matched to the k tech companies located in the same area, by contradiction.

Let us assume that there is a stable matching M for this problem instance where there is a student S_b who lives in the Bay area and is matched to a company C_o outside of the Bay area for his summer internship. As there are an equal number k of students and tech companies in the Bay area, this implies that M also includes a pair, where a tech company C_b, located in the Bay area is paired with a student S_o, who lives outside the Bay area. The pair (C_b, S_b) forms an unstable pair with respect to M, as each prefers the other relative to their partner in M.

This completes the proof.

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

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

$$n^{2.5},\ \sqrt{2n},\ n\ \log n$$

Answer:
The correct ordering will be:

$$\sqrt{2n} < n\ \log n < n^{2.5}$$

i.e., $\sqrt{2n} = O(n\ \log n)$ and $n\ \log n = O(n^{2.5})$.

We will prove each in turn.

$$\lim_{n\to\infty} \frac{\sqrt{2n}}{n\ \log n} = \lim_{n\to\infty} \frac{\sqrt{2}}{\sqrt{n}\log n} = 0$$

This proves $\sqrt{2n} = O(n\ \log n)$.

$$\lim_{n\to\infty} \frac{n\ \log n}{n^{2.5}} = \lim_{n\to\infty} \frac{\log n}{n^{1.5}} =$$

Using L'Hospital's rule, we have:

$$\lim_{n\to\infty} \frac{\log n}{n^{1.5}} = \lim_{n\to\infty} \frac{1/n}{1.5n^{0.5}} = \lim_{n\to\infty} \frac{1}{1.5n^{1.5}} = 0$$

This proves $n\ \log n = O(n^{2.5})$.

(b) Prove or disprove the following statement:

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

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

Answer:
False.

Take $f(n) = 2^n$. Then $g(n) = (f(\sqrt{n}))^2 = 2^{2\sqrt{n}}$. Clearly $\lim_{n\to\infty} f(n)/g(n) = 2^{n-2\sqrt{n}} = \infty$.

3. (5 points) Recurrence Relations

Derive an asymptotically tight bound for the following recurrence. Assume that $T(n)$ is $\Theta(1)$ for $n \leq 3$.

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

Answer:
In the recursion tree, each level contributes n until level $\log_3 n$. There are at most $\log_{3/2} n$ levels. So we obtain $T(n) \leq n \log_{3/2} n$ and at least $n \log_3 n$. So $T(n) = \Theta(n \log n)$.

Problem 4. (6 points) 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 $B[j] = A[(j+i) mod n]$ 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).

Algorithm 1: Find the Smallest Element in a Zig-Zag Array

```
1  l = 0; r = n − 1
2  while r − l ≥ 0 do
3  |  m = ⌊l + r/2⌋
4  |  if A[m] > A[l] then
5  |  |  l = m
6  |  else
7  |  |  r = m + 1
8  |  end
9  end
10 return A[(l + 1) mod n]
```

The algorithm above reduces the search span $(r - l)$ by half each time through the while loop and does some constant amount of work each time through the loop. The recurrence relation that corresponds to the algorithm can be expressed as $T(n) = T(n/2) + \Theta(1)$, which is familiar from Binary search and has the solution $T(n) = \Theta(\log\ n)$.

Problem 5. (8 points) Multimerge

Suppose we are given k sorted lists, each consisting of n elements. The *multimerge* problem is to merge the k lists into a single sorted list of length nk.

Design an efficient algorithm for the multimerge problem. Analyze the running time of your algorithm. The more efficient your algorithm is in terms of its worst-case running time, the more credit you will get.

Answer:
One simple algorithm is to repeatedly apply the merge operation as follows. First merge two lists of length n each to obtain a new sorted list of length $2n$; then merge this list with another list of length n to obtain a sorted list of length $3n$, and so on. Thus, in the ith iteration, merge a list of length in with a list of length n to obtain a sorted list of $(i + 1)n$. Running for $k - 1$ iterations yields the final sorted list of length nk.

For the first merge, the running time is $2n$, for the second merge it is $3n$, and so on. So the total running time is

$$2n + 3n + 4n + \ldots + kn = n(2 + 3 + \ldots + k) = n(k^2/2 + k/2 - 1) = \Theta(nk^2).$$

Here is a more efficient divide-and-conquer algorithm. We split the k lists into two parts, one containing $\lfloor k/2 \rfloor$ lists and the other containing $\lceil k/2 \rceil$ lists. Then, we recursively call the algorithm on the two parts and obtain two sorted lists, the total number of elements being nk. Finally, we merge the two lists in $O(nk)$ time using the merge algorithm covered in class to obtain a single sorted list with nk elements. The correctness of the algorithm directly follows from the correctness of the merge algorithm.

The recurrence for the worst-case running time is calculated as follows. Let $T(k)$ denote the worst-case running time of merging together k sorted lists, each of length n. Clearly, $T(1) = \Theta(1)$. And the recursion yields:

$$T(k) = 2T(k/2) + nk.$$

By unrolling the recurrence, we get

$$T(k) = nk + nk + ...(\lg k \text{ times }) = nk \lg k.$$