CPSC 320 Sample Midterm 1
January 2014

[12] 1. Short Answers

[3] a. Why is it useful to know that the furthest-in-the-future cache maintenance algorithm is optimal, even though it can not be implemented in practice?

Solution : It is because we can then evaluate other cache maintenance algorithms by comparing their performance to that of the furthest-in-the-future (optimal) algorithm.

[3] b. Mr. Isulo, the well-known alien computer scientist, draws a decision tree for the `MarvinSort` algorithm for $n = 10$ and shows it to you. What information can you infer from this tree (even though you have never heard of `MarvinSort`)?

Solution : You can determine every sequence of comparisons that the `MarvinSort` algorithm might perform on an array with 10 elements (including therefore the worst-case running time on this array size).

[3] c. Is the Gale-Shapley algorithm greedy? Explain why or why not.

Solution : It has some aspects typically associated with greedy algorithms (each woman proposes to the next man in her list without worrying about future events). However it also does things that greedy algorithm normally don't (engagements can be broken).
So either answer can be justified.

[3] d. Is $n/100 \in o(n)$? Explain why or why not.

Solution : No, it is not. The simplest way to verify this is to notice that the limit $\lim_{n\to\infty}(n/100)/n$ is a positive real number (to be specific, 0.01). Equivalently, observe that $n/100$ is not $\leq cn$ for any value $c < 0.01$.

[5] 2. A computer scientist who believes both genders are created equal wants to modify the Gale-Shapley algorithm as follows:

- In each iteration, either a free man or a free woman proposes to the first person on his/her preference list that he/she has not yet proposed to.
- The algorithm terminates when no free man or woman remains.

Give an example that proves that the matching produced by this modified version of the algorithm is not necessarily stable.

Solution : Suppose we have the following preference lists:

$$\begin{array}{ll} w_1: & m_1,\ m_2 \\ w_2: & m_1,\ m_2 \\ m_1: & w_2,\ w_1 \\ m_2: & w_2,\ w_1 \end{array}$$

and that w_1 first proposes to m_1, followed by m_2 proposing to w_2. At that point all women and men are engaged, and the algorithm stops. However this is not a stable matching, since m_1 and w_2 like each other better than their current partners.

[6] 3. Write a recurrence relation that describes the worst-case running time of the following algorithm as a function of n. You may ignore floors and ceilings. Note: I do not believe that this algorithm computes anything useful, so don't waste any time trying to understand what it does.

```
Algorithm BugsB(A, first, n)
    if n < 3 then
        return A[first] - 1
    endif

    x ← BinarySearch(A, first, n, BugsB(A, first + ⌊n/3⌋, ⌊n/3⌋))
    ElmerJ(A, first, n, x)
    return BugsB(A, first, ⌊n/6⌋) * BugsB(A, first + ⌊n/2⌋, ⌊n/4⌋)
```

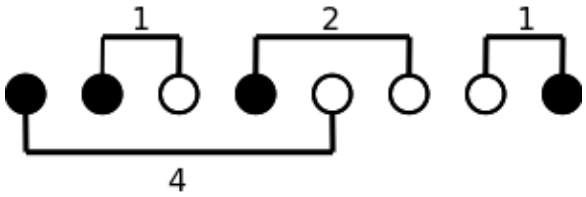
Assume that the call `ElmerJ(A, first, n, x)` runs in $\Theta(n \log n)$ time.

Solution :

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < 3 \\ T(\lfloor n/3 \rfloor) + T(\lfloor n/4 \rfloor) + T(\lfloor n/6 \rfloor) + \Theta(n \log n) & \text{if } n \geq 4 \end{cases}$$

[11] 4. Consider the following problem: there are $2n$ dots that are equally spaced on a line. Each dot is either white or black, and there are n dots of each color. The dots of a given color are not necessarily together; colors are interleaved arbitrarily. For instance:

We want to match each black dot with a white dot, while minimizing the sum of the distances between the elements of each pair. For instance, if we pair the dots of the previous figure as follows:



then the sum of the distances is $4 + 1 + 2 + 1 = 8$.

[8] a. Describe a greedy algorithm to find the matching that minimizes the sum of the distances between the elements of each pair. The input to your algorithm should be an array `C` of colors; for the example we would have `C[0] = black`, `C[1] = black`, `C[2] = white`, etc.

Solution : Here is one possible algorithm. We look at the dots from left to right, and maintain a stack S that contains the dots we have seen so far. When we look at a dot d:

- If S is empty, then we push d onto S.
- Else, if the dot on top of S has the same color as d then we push d onto S.
- Otherwise we pop the dot d' from the top of the stack, and pair d with d'.

[3] b. Analyze the time complexity of your algorithm from part (a).

Solution : If the dots were already sorted from left to right, then my algorithm runs in $O(n)$ time. If not, then it requires $O(n \log n)$ time as we need to sort the dots first.

[6] 5. In class, we proved the following theorem about the furthest in the future algorithm:

> If S_j is a reduced schedule that is the same as S_{FF} for the first j requests, then there is a reduced schedule S_{j+1} that is the same as S_{FF} for the first $j+1$ requests, and incurs no more misses than S_j in total.

[3] a. Does the schedule S_j need to be optimal for this theorem to work? If so, why? If not, what happens if S_j is not optimal?

Solution : No, S_j does not need to be optimal. If S_j is not optimal, then it is possible that S_{j+1} will be better (have fewer cache misses) than S_j.

[3] b. In the third case of the proof, we obtained S_{j+1} by first changing the element that S_j was evicting in response to the $(j+1)^{\text{st}}$ request. Why could we not then write that S_{j+1} would make the same decisions as S_j from request $j+2$ onwards (and stop with that statement)?

Solution : Because at that point, the caches of S_j and S_{j+1} did not contain exactly the same items.