CPSC 320 Midterm 1
Thursday February 6th, 2014

[13] 1. Short Answers

[3] a. In the version of the Gale-Shapley stable matching algorithm where women propose, is it possible for every man to get his worst possible partner? If so, what preference lists would result in that (unfortunate) outcome?

Solution : Yes, it is. It suffices for every woman to have a different first choice, and for each woman's first choice to be a man who ranks her last.

[3] b. What does each node of a decision tree represent?

Solution : A node represents all of the answers that are compatible with the results of the comparisons performed on the path from the root of the tree to that node (which corresponds to a possible sequence comparisons made by the algorithm).

[3] c. In case 3 of the proof of the theorem that we later used to show that the greedy caching strategy is optimal, we assumed that S_{FF} evicts an element e from the cache, whereas S_j ejects a different element f. We then stated that one of three things **must** happen before a request for e comes along:

- A request for f, where S_j evicts e.
- A request for f, where S_j evicts an element other than e.
- A request for some other element x, where S_j evicts e.

Why must one of these situations occur before any request for e?

Solution : Because e is the element that is accessed again furthest in the future. So f will be requested before e (assuming e is ever requested again).

[4] d. Consider the following two functions from $\mathbf{N}$ into $\mathbf{R}^+$:

$$f(n) = \begin{cases} 4n^3 & \text{if } n \text{ is even} \\ n^2/5 & \text{if } n \text{ is odd} \end{cases} \qquad g(n) = \begin{cases} n^3/6 & \text{if } n \text{ is even} \\ 2n^2 & \text{if } n \text{ is odd} \end{cases}$$

What is the relationship between f and g in terms of asymptotic notation? That is, is $f \in o(g)$, $f \in O(g)$, $f \in \Theta(g)$, $f \in \Omega(g)$, $f \in \omega(g)$, several of these, or none of them? Justify your answer.

Solution : Because $f(n) \leq 24g(n)$ for every $n \geq 1$, we know that $f \in O(g)$. Also, $f(n) \geq 0.1g(n)$ for every $n \geq 1$, which means that $f \in \Omega(g)$. Therefore $f \in O(g)$, $f \in \Omega(g)$ and $f \in \Theta(g)$.

[5] 2. Consider a problem $\mathcal{P}$ where, for an input of size n, there are 4^n possible answers. Prove as good a lower bound as possible on the worst-case running time of an arbitrary comparison-based algorithm for $\mathcal{P}$.

Solution: Every comparison-based algorithm for $\mathcal{P}$ can be represented by a family of decision trees T_1, T_2, T_3, ... Tree T_n will have 4^n leaves, which means that $4^n \leq 2^h$, where h is the height of T_n. Thus $h \geq \log_2 4^n$. Because $\log_2 4^n = 2\log_2 2^n = 2n$, this means that $h \geq 2n$. Therefore every comparison-based algorithm for $\mathcal{P}$ performs in $\Omega(n)$ time comparisons.

[5] 3. Ms. Ejiul, an alien computer scientist, designed a greedy algorithm that solves a graph problem by adding vertices and edges to her solution one at a time. She then proved that given a (not necessarily optimal) solution H that adds the same first j vertices and edges as her greedy algorithm, she can modify it to obtain a solution H' that is at least as good as H, and adds the same first $j+1$ vertices and edges as her greedy algorithm.

Is Ms. Ejiul's greedy algorithm optimal? Why or why not?

Solution: Yes, her algorithm is optimal. The proof is exactly the same as for the optimal caching algorithm discussed in class: if O_1 is an optimal solution, then we can obtain a succession O_1, O_2, O_3, ..., O_t of solutions where O_j adds the same j vertices and edges as Ms. Ejiul's greedy algorithm, and for very j, O_{j+1} is at least as good as O_j (meaning it's also optimal). This implies that O_t (which adds exactly the same vertices and edges as the greedy solution, and hence **is** the greedy solution) is also optimal.

[8] 4. Recurrence relations

[4] a. 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 ComputeTwo(A, first, n)
prod ← 1
if n ≥ 25 then
  for i ← 1 to $\lfloor\sqrt{n}\rfloor$ do
    n ← n - 5
    prod ← prod * ComputeTwo(A, first, n)
  endfor
endif
return prod
```

Solution:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < 25 \\ \Theta(\lfloor\sqrt{n}\rfloor) + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} T(n-5i) & \text{if } n \geq 25 \end{cases}$$

[4] b. Write an algorithm whose running time can be described by the recurrence:

$$T(n) = \begin{cases} T(n/5) + 2T(n/4) + \Theta(n) & \text{if } n \geq 5 \\ \Theta(1) & \text{if } n < 5 \end{cases}$$

Your algorithm does not need to compute anything meaningful.

Solution : There are many possible algorithms. Here is a simple one:

```
Algorithm ComputeOne(A, first, n)

sum ← 0
if (n ≥ 5) then
    for i ← 0 to n-1 do
        sum ← sum + A[i]
    endfor

    sum ← sum + ComputeOne(A, first, n/5)
    sum ← sum + ComputeOne(A, first, n/4)
    sum ← sum + ComputeOne(A, first, n/4)
endif

return sum
```

[14] 5. You and a group of your friends want to play tug-of-war (two teams pulling on opposites sides of a rope). Being the only computer scientist in the group, you have been asked to design an algorithm to build two teams A and B that are as equal as possible. You formalize the problem as follows:

- Each person i has a strength s_i.
- You would like the sum of the strengths of the people on team A to be as close to equal as possible to the sum of the strengths of the people on team B. That is, you want to minimize

$$\left| \sum_{i \in A} s_i - \sum_{j \in B} s_j \right|$$

Note that the two teams do not need to have the same number of people.

[8] a. Describe a greedy algorithm to build the teams A and B. Your algorithm does not need to always succeed at minimizing the difference in total strength between the two teams, but it should make a good attempt at it.

Solution : Here is one possible algorithm:

```
Algorithm buildTeams(S)
  sort S by decreasing strength
  strengthA ← 0
  strengthB ← 0
  for i ← 0 to length[S] - 1 do
    if strengthA < strengthB then
      add person i to A
      strengthA ← strengthA + strength[i]
    else
      add person i to B
      strengthB ← strengthB + strength[i]
    endif
  endfor
  return A, B
```

[3] b. Analyze the time complexity of your algorithm from part (a). Specify only as much of your implementation (for instance, data structures) as needed for your analysis. There is no need to get the most efficient possible implementation of your algorithm, but excessively pessimistic analyses (e.g. $O(n^4)$ when an algorithm could be implemented to run in $O(n \log n)$ time) will not get full marks.

Solution : My algorithm runs in $O(n \log n)$ time, because of the sorting step.

[3] c. Give an example where your algorithm will not return the optimal solution (the one that minimizes the strength difference between the two teams).

Solution : Suppose we have six people with strengths 10, 9, 6, 5, 4 and 2. The algorithm from part (a) will make two teams with strengths $10 + 5 + 2 = 17$ and $9 + 6 + 4 = 19$. However there is a solution where both total strengths are equal: $10 + 6 + 2 = 9 + 5 + 4 = 18$.