CS-310, Fall 2021
Assignment 1
Assigned: Oct. 2, Due: Monday, October. 11, 11:55PM

October 1, 2021

Total Points = 150

Problem 1. (10 points) Gale and Shapely published their paper on the Stable Matching Problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

Basically, the situation wa the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than were slots available in the m hospitals.

The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

We say that an assignment of students is *stable* if neither of the following situations arise.

1. First type of instability: There are students s and s' and a hospital h, so that
   - s assigned to h, and
   - s' is assigned to no hospital, and
   - h prefers s' to s.
2. Second type of instability: There are students s and s' and hospitals h and h', so that
   - s assigned to h, and
   - s' is assigned h', and
   - h prefers s' to s, and
   - s' prefers h to h',

So, we basically have the Stable Matching Problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals, and give an algorithm to find one.

Problem 2. (18 points) Suppose you have algorithms with the six running times listed below. (Assume these ar the exact number of operations performed as a function of the input size n.) Suppose you have a computed that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

a) n^2

b) n^3

c) $100n^2$

d) $nlogn$

e) 2^n

f) 2^{2^n}

Problem 3. (5 points) Show that 2^{n+1} is $O(2^n)$

Problem 4. (15 points, 3 points for each case) In each case below, demonstrate whether the given expression is true or false.You need to provide proper reason to receive credit.

a) $5n^3 \in O(n^3)$

b) $100n^2 \in O(n^4)$

c) $\log n^2 \in O(\log n)$

d) $(n^2 + 7n - 10)^3 \in O(n^6)$

e) $\sqrt{n} \in O((\log n)^3)$

Problem 5. (18 points, 2 points for each row) In each row of the following table, write whether $f = O(g)$, or $f = \Omega(g)$, or both i.e. $f = \Theta(g)$

$f(n)$	$g(n)$	Your Answer
100	17	
10^{200}	$n - 10^{200}$	
$n^2 \log 300$	$n \log n$	
n^{100}	2^n	
$n \log n$	$n - 100$	
$\sqrt{n}$	$\log n$	
$n^{1.01}$	$n + 100$	
$2 \log n$	$\log(n^2)$	
$\log(n^2)$	$(\log n)^2$	

Problem 6. (18 points) Order the following functions of n from the smallest to largest complexity. If two have the same complexity put them in one line next to each other.

$$n \log n, \quad (n-5)(n), \quad 10 \log 300, \quad n + 7 \log(n^2),$$
$$2^n, \quad 10n^2 + 15n, \quad 17, \quad 15n^3 + n \log n, \quad \frac{3n^8}{n^6}$$
$$10^{200}, \quad n^2 \log 300, \quad 2 \log_{10} 10^{200}, \quad n^2 + 7 \log(n^2),$$
$$3^{n^2}, \quad n^2 + \sqrt{n}, \quad n^2 \log n, \quad n^4 + n^2 + n^2 \log n, \quad n^4$$

Problem 7. [Complexity Classes] (18 points, 2 points for each function) Order the following functions of n from the smallest to largest complexity and also identify the complexity class for each. If two have the same complexity put them in one line next to each other. Briefly explain next to each line why these functions are in the same complexity class.

$$10^{200}, \quad n^2 \log 300, \quad 2 \log_{10} 10^{200}, \quad n^2 + 7 \log(n^2),$$
$$3^{n^2}, \quad n^2 + \sqrt{n}, \quad n^2 \log n, \quad n^4 + n^2 + n^2 \log n, \quad n^4$$

Problem 8. (24 points, 3 points for each part) Let $f(n)$ and $g(n)$ by asymptotically positive functions. Prove or disprove each of the following conjectures.

1. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.
2. $f(n) + g(n) = \Theta(\max(f(n), g(n)))$.
3. $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.
4. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.
5. $f(n) = O((f(n))^2)$.
6. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.
7. $f(n) = \Theta(f(n/2))$.
8. $f(n) + \Omega(f(n)) = \Theta(f(n))$.

Problem 9. (24 points - 5 + 5 + 14) Consider the following basic problem. You are given an array A consisting of n integers $A[1], A[2], \ldots A[n]$. You would like to output a two-dimensional $n - by - n$ array B in which $B[i, j]$ (for $i < j$) contains the sum of array enries $A[i]$ through $A[j]$ - that is, the sum $A[1] + A[2], + \cdots + A[n]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it does not matter what is output for these values.)

Following is a simple algorithm to solve this problem.

Algorithm 1 : Summing array entries

```
for i = 1 to n − 1 do
    for j = i + 1 to n do
        B[i, j] ← Σ_{k=i}^{j} A[k]
return B
```

1. For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm)

2. For this same function f, show the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptoticaly tight bound of $\Theta(f(n))$ on the running time.

3. Although the algorithm given above is the most natural way to solve the problem - after all, it just iterates through the relevant entries of the array B, filling in a value for each – it contains some highly unnecessary sources of inefficiency. Give a different algorith to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.