Homework 1 – Due April 21st, 2017 on Canvas

Homework Guidelines

Please make sure you read the collaboration policy, and write the following as the first line of your homework: "I have read and agree to the collaboration policy. ⟨`Your name`⟩." Your homework will not be graded if you do not write this.

Collaboration policy Collaboration on homework problems is permitted, but not encouraged. You are allowed to collaborate with *at most two students enrolled in the class*. You must mention the name of your collaborators clearly on the first page of your submission. Even if you collaborate, you are *expected to write and submit your own solution independent of others*, and your collaboration should be restricted to discussions only. Also, you should be able to explain your solution verbally to the course staff if required to do so. *Collaborating with any one not enrolled in the class, or taking help from any online resources for the homework problems is strictly forbidden.*

The Computer Science Department of UCSC has a zero tolerance policy for any incident of academic dishonesty. If cheating occurs, consequences within the context of the course may range from getting zero on a particular assignment, to failing the course. In addition, every case of academic dishonesty will be referred to the student's college Provost, who sets in motion an official disciplinary process. Cheating in any part of the course may lead to failing the course and suspension or dismissal from the university.

How to submit your solutions Each problem must be **typed** up separately (in at least an 11-point font) and submitted in the appropriate assignment box on the Canvas website as a PDF file.

This means that you will submit 4 separate files on Canvas, one for each problem!

You are strongly encouraged, but not required, to format your problem solutions in LaTeX. Template HW files and other LaTeXresources are posted on the course webpage. LaTeXis a free, open-source scientific document preparation system. Most technical publications in CS are prepared using this tool.

You might want to acquire a LaTeX manual or find a good online source for LaTeX documentation.

The top of each problem should include the following:

- your name,
- the acknowledgement to the collaboration policy,
- Your choice of *homework heavy versus homework light grading*,
- the names of all people you worked with on the problem (see the handout "Collaboration and Honesty Policy"), indicating for each person whether you gave help, received help or worked something out together, or "Collaborators: none" if you solved the problem completely alone.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. Understandability of your answer is as desirable as correctness, because communication of technical material is an important skill. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

Assigned Problems

Exercises (Do not hand in) Chapter 1, Problems 1-3, 5. Chapter 2, Problems 3-5, 7.

Following are the problems to be handed in, 25 points each.

1. (**Resident Matching**, 2-page limit – your solutions should fit on two sides of 1 page).

   The situation is the following. There were m teams at Google, each with a certain number of available positions for hiring interns. There were n students who want internships at Google this summer, each interested in joining one of the teams. Each team had a ranking of the students in order of preference, and each student had a ranking of the teams in order of preference. We will assume that there were more students who want an internship at Google than there were slots available in the m teams.

   The interest, naturally, was in finding a way of assigning each student to at most one team at Google, in such a way that all available positions in all teams were filled. (Since we are assuming a surplus of potential interns, there would be some students who do not get assigned to any team.) We say that an assignment of students to Google teams is stable if neither of the following situations arises.

   - The first type of instability that can occur is that there is a team t, and there are students s and $s\prime$, so that
     - s is matched with t, and
     - $s\prime$ is assigned to no team, and
     - t favors $s\prime$ over s.
   - The second type of instability that can occur is that there are teams t and $t\prime$ and students s and $s\prime$ so that
     - s is matched with t, and
     - $s\prime$ is matched with $t\prime$, and

- t favors $s\prime$ over s, and
- $s\prime$ favors t over $t\prime$.

So we basically have the Stable Matching Problem, except that, one, teams generally want more than one intern, and, two, there is a surplus of students who want internships at Google. Show that there is always a stable assignment of students to Google teams, and give an algorithm to find one.

Please give a clear description of your algorithm. Don't forget to prove its correctness and analyze its time and space complexity.

2. (**Time complexity**, 2-page limit – your solutions should fit on two sides of 1 page). Part (a) has 15 points and part (b) has 10 points. The top of your solution for part (a) should have the functions in order by their letter, with no spaces, commas, etc. between them. (For example, abc). If you do not include this you will automatically lose 75% of the credit. (Functions that are equivalent should be in alphabetical order)

   (a) Rank the following functions by increasing order of growth, that is, find an arrangement $g_1, ...$ of the functions satisfying $g_1(n) = O(g_2(n)), g_2(n) = O(g_3(n)), ...$ Break the functions into equivalence classes so that f and g are in the same class if and only if $f(n) = \Theta(g(n))$. Note that $\log(\cdot)$ is the base 2 logarithm, $\log_b(\cdot)$ is the base b logarithm, $\ln(\cdot)$ is the natural logarithm, and $\log^c(n)$ denotes $(\log(n))^c$ (for example, $\log^2(n) = \log(n) \times \log(n)$).

a. $\ln(\ln n)$	**b**. $n \log n$	**c**. $14 \log_3 n$	**d**. $\sum_{i=5}^{n} \frac{(i+1)}{2}$	**e**. $\log^2(n)$
f. n^2	**g**. $\sum_{i=1}^{n} \left(\frac{1}{2}\right)^i$	**h**. $\log(n!)$	**i**. 3^n	**j**. $n^{\log 7}$
k. $\sum_{i=1}^{n} 3^i$	**l**. $2^{\log^2(n)}$	**m**. $2^{\log n}$	**n**. $n!$	**o**. n
p. $2^{\log_4 n}$	**q**. $\sqrt{n}$	**r**. $\log(n^2)$	**s**. $4^{\log n}$	**t**. $(\frac{5}{4})^n$

   (b) For each of the following statements, decide whether it is always true, never true, or sometimes true for asymptotically nonnegative functions f and g. If it is always true or never true, give a proof. If it is sometimes true, give one example for which it is true, and one for which it is false.

   i. $f(n) + g(n) = \Omega(\max(f(n), g(n)))$
   ii. $f(n) = \omega(g(n))$ and $f(n) = O(g(n))$
   iii. Either $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$ or both.

3. (**Induction**, 2-page limit – your solutions should fit on two sides of 1 page). Part (a) has 10 points and part (b) has 15 points.)

   (a) (**Uniform shuffling**) Let $A[1, \cdots, n]$ be an array of integers. A uniform shuffle of A is a set of n random elements from A (without replacement), such that the probability of selecting any such set is the same. Consider the following algorithm to generate a uniform random shuffle:

UNIFORMSHUFFLE(A)

```
1  for i ← n downto 1
2      do j ← random integer such that 1 ≤ j ≤ i
3         exchange A[i] and A[j]
4  return A
```

Prove that the algorithm indeed generates a uniform random shuffle of A. What is the running time of the algorithm, given that generating random integer takes time $O(1)$? *Hint: Start by thinking of what a uniform shuffle means in terms of probability.*

(b) Point out the error in the following proof by induction.

Claim: Given any set of b buses, all buses lead to the same destination.

Proof: We proceed by induction on the number of buses, b.

Base case: If $b = 1$, then there is only one bus in the set, and so all buses in the set lead to the same destination.

Induction step: For $k \geq 1$, we assume that the claim holds for $b = k$ and prove that it is true for $b = k + 1$. Take any set B of $b + 1$ buses. To show that all buses lead to the same destination, we take the following approach. Remove one bus from this set to obtain the set B_1 with just b buses. By the induction hypothesis, all the buses in B_1 lead to the same destination. Now go back to the original set and remove a different bus to obtain a the set B_2. By the same argument, all the buses in B_2 lead to the same destination. Therefore all the buses in $B = B_1 \cup B_2$ must lead to the same destination, and the proof is complete.

4. (**Divide and Conquer**, 2-page limit – your solutions should fit on two sides of 1 page).

After dating for several years, Jack and Anthony have finally decided to move in together. As part of this process, each of them wants to bring his n alphabetically sorted books over to the new place. Due to some weird reason, they want to find out who owns the median book of the joint book collection, which has $2n$ books. In this joint book collection, the median would be the n-th book among the union of the $2n$ alphabetically sorted books.

Because their original book collections are already sorted, they manage to find out who owns the median in $\Theta(\log n)$. They did not have to reorder the joint book collection, but rather it was enough for them to just query individual values from their original book collections. What algorithm did they use? Prove that this algorithm is correct. Find the recurrence relation and show that it resolves to $\Theta(\log n)$.