HW 1 (due Wednesday, at noon, September 5, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.1**

This homework contains three problems. **Read the instructions for submitting homework on the course webpage**.

You also should do quiz 0 online (on moodle).

Collaboration Policy: For this homework, and all following homeworks, you can submit in groups of size up to 3. You are **strongly encouraged** to submit/work in groups – you will do worse in the course if you work on your own. Really.

Read the course policies before starting the homework.

- Homework 0 and Quiz 0 test your familiarity with prerequisite material: big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction, to help you identify gaps in your background knowledge. You are responsible for filling those gaps. The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.
- Please carefully read the course policies on the course web site. If you have any questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email. In particular:
  - **Submission**: Please submit the solution to each question in a separated PDF file uploaded to gradescope. Have your name and NetIDs clearly printed on first page.
  - **You may use any source at your disposal**: paper, internet, electronic, human, or other, but you must write your solutions in your own words, and you must **cite explicitly**[1] every source that you use (except for official course materials [and even if you use course material, you might want to give a ref]). Please see the academic integrity policy for more details.
  - No late homework will be accepted for any reason. However, we may forgive quizzes or homeworks in extenuating circumstances; ask the instructor for details.
  - Answering "I don't know" to any (non-extra-credit) problem or subproblem, on any homework or exam, is worth 25% partial credit. Such credit would be capped at 10% for exams. You will not even remotely pass the class if you answer IDK on all questions in homeworks/exams.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this process for all n", instead of an explicit loop, recursion, or induction, will receive a score of 0.
  - You would lose **all points** for unnecessarily long solutions (say longer than twice what the instructor considers reasonable length). A long correct solution is as useless as a short, brilliant incorrect solution.
  - In particular, partial credit would be given to work that has real merit. Just writing stuff would not get you points in this class. If you do not have a clue, use IDK.
  - Unless explicitly stated otherwise, every homework problem requires a proof.

[1] For example: "I found the solution to this exercise on `http://www.endoftheinternet.com/`. Since I understand the submission guidelines, I read this solution carefully, understood it, believe that it is correct, and I wrote it out in my own words. I was, of course, not so mind boggling stupid to just cut and paste some random text I found on the internet, because I know the class staff is advanced enough that they can also search my solution and see if I copied it from somewhere." **(Of course, you need only the first sentence.)**

1 (100 PTS.) How to solve a recurrence?

Solving recurrences is one of the dark arts of computer science, made dark by the awful way it is being taught. This exercise takes you through the process of solving many such recurrences. Once you master the technique shown in this exercise, it should be enough for you to be able to solve all the recurrences you would encounter in this class. Jeff Erickson has class notes on how to solve recurrences, see here:

http://jeffe.cs.illinois.edu/teaching/algorithms/notes/99-recurrences.pdf.

(Specifically, section 3.)

Also, see Part IV, and also the detailed solution of the merge sort algorithm recurrence, in the following:

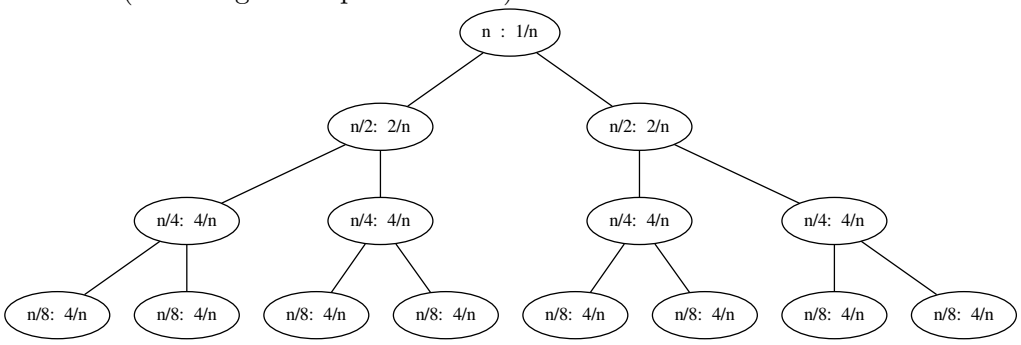
https://courses.engr.illinois.edu/cs374/fa2017/slides/10_recursion_divide_conquer.pdf

For each of the following recurrences, provide the following information:

(I) d: The depth of the recurrence tree.
(II) n_i: The number of nodes in the recurrence tree of depth exactly i.
(III) L_i: The total contribution associated with all the nodes of level i (potentially an upper bound, as tight as possible, on this quantity).
(IV) An upper bound, as tight as possible, on $\Delta = \sum_{i=0}^{d} L_i$.
(V) An upper bound, as tight as possible, on the recurrence value.

(Provide detailed proof if the calculations/details are not immediate.) For all the recurrences below, the assumption is that the function is bounded by a small positive constant, if the parameter n is smaller than, say, 10,

Example: $f(n) = 2f(\lfloor n/2 \rfloor) + 1/n$.
Solution:
(I) The recurrence tree in this case is a full binary tree. Here is a figure showing the first four levels (assuming n is a power of two):



Since n can be divided at most $\lceil \log_2 n \rceil$ before it becomes smaller than 1, it follows that the depth of the recurrence is $d = \lceil \lg n \rceil$ (reminder: $\lg n = \log_2 n$).
(II) $n_i \leq 2^i$ – since the ith level of a binary tree has 2^i nodes.
(III) $L_i = n_i \cdot 1/(n/2^i) \leq 2^{2i}/n$.
(IV) $\Delta = \sum_{i=0}^{d} L_i = \sum_{i=0}^{d} 2^{2i}/n = O(n)$.
(V) $f(n) = O(\Delta) = O(n)$.

1.A. (10 PTS.) $A(n) = A\Big(\lceil \sqrt{n}/3 \rceil + \lfloor \log n \rfloor\Big) + n$.

1.B. (10 PTS.) $B(n) = 3B\Big(\lceil n/2 \rceil - 5\Big) + n/\log n$.

Hint: You might want to first solve the easier recurrence $B'(n) = 3B'\Big(\lceil n/2 \rceil - 5\Big) + n$.

1.C. (10 PTS.) $C(n) = C(\lfloor 3n/4 \rfloor) + \sqrt{n}$.

1.D. (10 PTS.) $D(n) = D(\lfloor \log n \rfloor) + \log n$.

1.E. (10 PTS.) $E(n) = n + \lfloor \sqrt{n}/2 \rfloor \cdot E(\lfloor \sqrt{n} \rfloor)$.

1.F. (10 PTS.) $F(n) = F(\lfloor n/4 \rfloor) + F(\lfloor n/5 \rfloor) + F(\lfloor n/6 \rfloor) + O(n)$.

1.G. (10 PTS.) $G(n) = G\Big(\lfloor (5/12)n \rfloor\Big) + G\Big(\lfloor n/3 \rfloor\Big) + G\Big(\lfloor n/4 \rfloor\Big) + O(n)$.

1.H. (10 PTS.) $H(n) = \sum_{i=1}^{4} H(N_i) + O(n^2)$, where $N_1, N_2, N_3, N_4 \leq n/2$.

For the following recurrences are different, and you need to provide only an upper bound as tight as possible on the recurrence solution (they do not fall immediately under the above framework). Prove your answer.

1.I. (10 PTS.) $I(n) = \min_{0<k<n} \big(3 + I(k) + I(n-k)\big)$.

1.J. (10 PTS.) $J(n) = \frac{n}{n-3} J(n-1) + 1$.

2 (100 PTS.) The best of all possible aliens.

The following exercise verify that you can do some basic probability calculations, how to compute expectations, manipulate them, and how to apply Markov's inequality. (Consult with the wikipedia page `https://en.wikipedia.org/wiki/Expected_value` if you want a refresh on this stuff.)

2.A. (20 PTS.) Let $v_1, \ldots, v_n$ be n aliens. Let $x_1, \ldots, x_n$ be a random permutation of $1, \ldots, n$ chosen uniformly at random. The alien v_i is a ***leader*** at time i, if $x_i < x_1, \ldots, x_{i-1}$. What is the probability of v_i to be a leader at time i?

2.B. (20 PTS.) Let X_i be an indicator variable that is 1 if v_i is a leader at time i (and zero otherwise). What is $\mathbb{E}[X_i]$ (as a function of i)?

2.C. (20 PTS.) Provide an upper bound, as tight as possible, on $\mathbb{E}[\sum_{i=1}^{n} X_i]$. Prove your answer.

2.D. (20 PTS.) Let $Y_i = -1$ with probability half, and $Y_i = +1$ with probability half. Let $Y = \sum_{i=1}^{n} Y_i$. What is $\mathbb{E}[Y]$, $\mathbb{E}\big[Y^2\big] = \mathbb{E}\Big[(Y_1 + Y_2 + \cdots + Y_n)^2\Big]$ and $\mathbb{E}\big[Y^3\big]$. Provide exact bounds, and prove your answer.

2.E. (20 PTS.) Arguing as above, one can show that $\mathbb{E}\big[Y^4\big] = 6n^2 - 5n$. Prove that, for any $t > 0$, we have $\mathbb{P}\Big[Y \geq t\sqrt{\sqrt{6}n}\Big] \leq 1/t^4$ (hint: Translate this event to an event on Y^4).

3 (100 PTS.) Some boolean logic.

The following exercise is intended to give you some intuition why being able to decide if a boolean formula can be used to solve various optimization problems.

A boolean formula is defined over variables that might have a value either 0 or 1. A basic token is either a variable, or 0 or 1. A formula is now defined recursively, as follows:

- Given two boolean formulas F_1, F_2, the *and* of the two formulas is $F_1 \wedge F_2$ – this is a new formula that is true if and only if both F_1 and F_2 are true.
- Given two boolean formulas F_1, F_2, the *or* of the two formulas is $F_1 \vee F_2$ – this is a new formula that is true if and only if at least one of F_1 and F_2 is true.
- Given a boolean formulas F, the *not* of F is $\overline{F}$ – this is a new formula that is true (i.e., 1) if and only if F is false (i.e., 0), and vice versa.
- Given a formula F, and a boolean variable y, the new formula $y = F$ is true if and only if y and F have the same value.

As an example $y = \overline{(y = x \wedge z)}$ is a valid boolean formula. An *input* variable is a variable whose value is specified in advance. A variable in a formula that is not an input variable, is a *free* variable. A formula is *satisfiable* if there is an assignment to its free variables such that the formula evaluates to true (for the specified values of the input variables).

In the following you can assume n is a power of 2.

3.A. (10 PTS.) Let $F_{k,n}(x_1, \ldots, x_n)$ be a boolean formula that is **satisfiable** if and only there are exactly k variables in the input variables that have value 1 (and the rest are 0). Provide a formula for $F_{n,n}(x_1, \ldots, x_n)$.

3.B. (10 PTS.) Provide the formula for $F_{0,n}(x_1, \ldots, x_n)$.

3.C. (20 PTS.) Let $n' < n$, and assume you are given formulas $F_{j,n'}$, for $j = 0, \ldots, n'$, provide using them a formula for $F_{k,n}(x_1, \ldots, x_n)$. (Again, the formula is satisfiable, if and only if there are exactly k ones in the input bits $x_1, \ldots, x_n$.)

3.D. (30 PTS.) Using (C) describe (in high level – no need for pseudo-code) an algorithm that computes a formula for $F_{k,n}(x_1, \ldots, x_n)$. What is the size of the formula (i.e.., the number of variables and operators if we write the formula explicitly), as function of k and n (provide a simple closed form upper bound). For credit the algorithm you describe should output a formula of size polynomial in n (independent of the value of k).

3.E. (30 PTS.) You are given a graph G over n vertices (the graph provided via an adjacency matrix M of size $n \times n$, where $M(i,j) = M(j,i) = 1 \iff$ the edge $ij \in \mathsf{E}(\mathsf{G})$), and a number k. Describe an algorithm that computes a formula F, which accepts as input the n^2 bits of M, such that the formula F is satisfiable $\iff$ there is a partition of the vertices of G into two sets S and $\overline{S} = \mathsf{V}(\mathsf{G}) \setminus S$, such that the cut $(S, \overline{S}) = \left\{ ij \in \mathsf{E}(\mathsf{G}) \;\middle|\; i \in S, j \in \overline{S} \right\}$ contains exactly k edges. What is the size of the computed formula (as usual, the formula has to be of polynomial size).