# HW 1

Due on **Wednesday, September 1, 2021 at 10am**

**CS 473: Algorithms, Fall 2021** Version: **1.1**

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

**Collaboration Policy:** For all 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 test your familiarity with prerequisite material: big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and 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 homeworks in extenuating circumstances; ask the instructor for details.
  - 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, just say so.
  - Unless explicitly stated otherwise, every homework problem requires a proof.
  - Submission of homeworks is via gradescope.
  - Every problem or subproblem solution **must** start on its own page, and contain the problem/subproblem number on the top of the page.

---

[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 some 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.

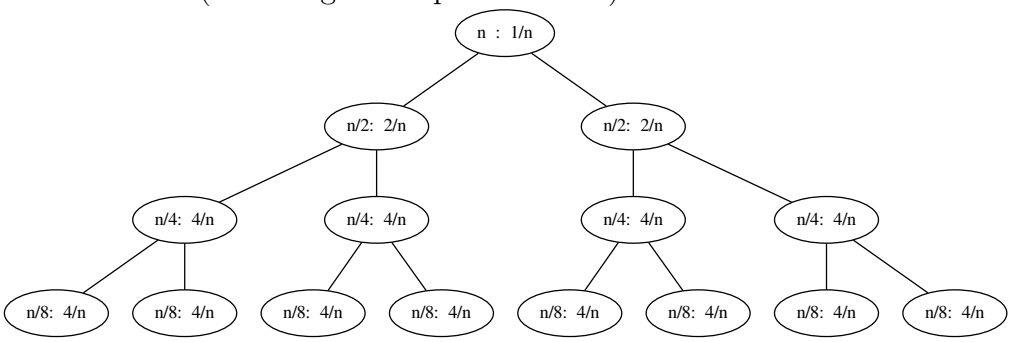
(Provide detailed proof if the calculations/details are not immediate.) For all the recurrences below, the assumption is that the function is bounded by a small positive constant, if the parameter $n$ is smaller than, say, 10,

[As tight as possible = within reasonable effort. Usually an estimate correct within a constant factor is good enough if you can get it. For the depth of the recurrence part, you want the exact quantity within small additive factor.]

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):




[The tree of course continues in this fashion all the way down.]

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 $i$th 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) $f(n) = O(\Delta) = O(n)$.

[Explanation (you do not need to include this in your solution): $f(n)$ can be interpreted as the total amount of work done in all the nodes of the tree, which is just $\Delta$. Or $O(\Delta)$ since constants don't matter.]

**1.A.** (10 PTS.) $A(n) = A\Big(\lceil n^{1/3} \rceil\Big) + 1$.

**1.B.** (10 PTS.) $B(n) = B\Big(\lceil n^{1/3}/3 \rceil + \lfloor \log n \rfloor\Big) + n$, where $\log n = \log_{10} n$.

[Hint: Solve for a simpler recurrence $B'(n)$, such that $B'(n) \geq B(n)$, and use part (A).]

**1.C.** (20 PTS.) $C(n) = 5C\Big(\lceil n/4 \rceil\Big) + n$.

**1.D.** (20 PTS.) $D(n) = D(\lfloor \log n \rfloor) + \log \log n$.

**1.E.** (20 PTS.) $E(n) = 2E\Big(\lfloor (5/24)n \rfloor\Big) + E\Big(\lfloor n/3 \rfloor\Big) + E\Big(\lfloor n/4 \rfloor\Big) + O(n)$.

[Hint: What is the total size of the subproblems in the first level of the recurrence? (Repeat for second level, and third level)]

**1.F.** (20 PTS.) $F(n) = \sum_{i=1}^{8} F(N_i) + O(n^3)$, where $N_1, \ldots, N_8 \leq n/2$.

## 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.** (30 PTS.) Let $v_1, \ldots, v_n$ be $n$ aliens. Let $\pi : [\![n]\!] \to n$ be a random permutation of $[\![n]\!] = \{1, \ldots, n\}$ chosen uniformly at random. Every alien $v_i$ has $f_i$ friends, and let $F(v_i) \subseteq \{v_1, \ldots, v_n\}$ be its set of friends. Thus, the align $v_i$ is placed at location $\pi(i)$ in the permutation. The alien $v_i$ is a ***leader*** at time $i$, if $\pi(i) < \min_{v_j \in F(v_i)} \pi(j)$. That is, $v_i$ (i.e., $i$) appears in the random permutation before all its friends. What is the probability of $v_i$ to be a leader?

**2.B.** (30 PTS.) Let $X_i$ be an indicator variable that is 1 if $v_i$ is a leader. What is $\mathbb{E}[X_i]$? Provide a concise formula for the quantity $\mu = \mathbb{E}[\sum_{i=1}^{n} X_i]$.

**2.C.** (40 PTS.) Provide a self contained proof that $\mu \geq n/\big((\sum_{i=1}^{n} f_i/n) + 1\big)$.
[Hint: Prove that for any $x_1, \ldots, x_n > 0$, we have that $\sum_i 1/x_i \geq n/(\sum x_i/n)$.]

## 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.

Build a collection $\mathcal{F}$ of formulas, that uses the input variables $x_1, \ldots, x_n$, and involve potentially more variables. In particular, the new formulas have also variables $y_1, \ldots, y_n$. Describe how to construct a family of formulas $\mathcal{F}$ such that, all the formulas in $\mathcal{F}$ are satisfiable simultaneously if and only if $y_1, \ldots, y_n$ are the bits $x_1, \ldots, x_n$ sorted in decreasing order (hint, for $x_1, x_2$, $\mathcal{F}(2) = \{y_1 = x_1 \vee x_2 \text{ and } y_2 = x_1 \wedge x_2\}$ works).

(Your construction should be simple and self contained.)

What is the total length of all the formulas in $\mathcal{F}$ as a function of $n$? (A construction with polynomial dependency on $n$ is good enough. Getting the "right" answer here [i.e., $O(n \log n)$] is doable but extremely difficult.)

(Naturally, asking for an assignment that satisfies all the formulas in a collection of formulas $\mathcal{F}$, is equivalent to asking for a satisfying assignment for the single formula $\bigwedge_{\phi \in \mathcal{F}} \phi$.)

[Hint: The easiest solution is probably trying to simulate bubble sort.]