HW 1

Due on **Wednesday, August 30, 2023 at 9pm**

CS 473: Algorithms, Fall 2023 Version: **2.0**

This homework contains three problems.

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 alone. Really.

Read the course policies before starting the homework.

- Homework 1 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 on EdStem, or in office hours.
- 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.

    Typing your solutions is strongly encouraged. Using LaTeXis encouraged.
  - **You may use any source at your disposal**: paper, internet, ChapGPT, LLMs, 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?

This exercise takes you through the process of solving some 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:

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 term.]

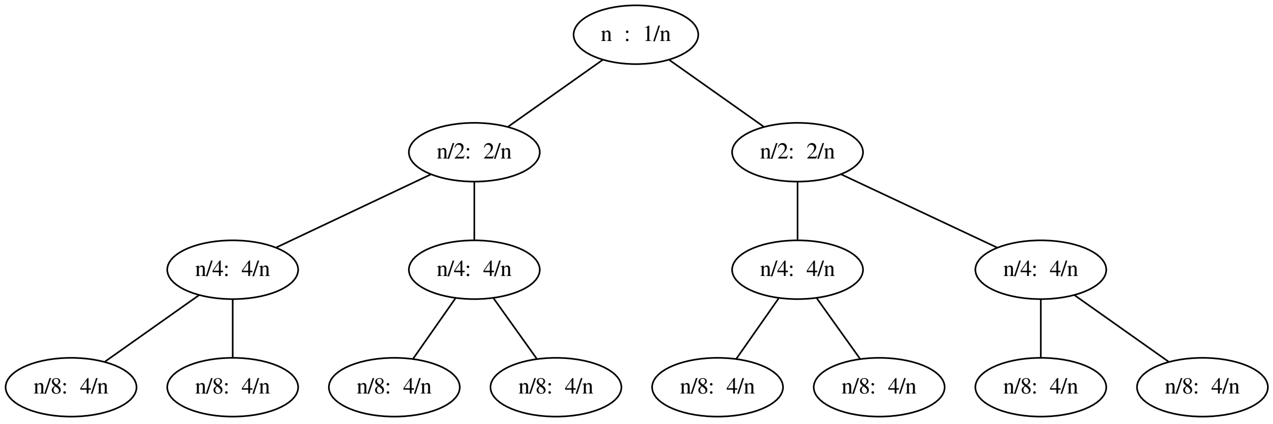

Figure 1:

Example: $f(n) = 2f(\lfloor n/2 \rfloor) + 1/n$.

Solution:

(I) The recurrence tree in this case is a full binary tree. See Figure 1 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 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) $f(n) = O(\Delta) = O(n)$.

[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 Δ. Or $O(\Delta)$ since constants don't matter.]

1.A. (25 PTS.) $A(n) = A\Big(\lceil \log_2 n \rceil\Big) + \lfloor \log_2 n \rfloor$.

(Hint: Iterated log.)

1.B. (25 PTS.) $B(n) = 6B\Big(\lceil n/5 \rceil\Big) + 2n$.

1.C. (25 PTS.) $C(n) = 2C\Big(\lfloor (1/12)n \rfloor\Big) + C\Big(\lfloor n/3 \rfloor\Big) + C\Big(\lfloor n/2 \rfloor\Big) + O(n)$.

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

1.D. (25 PTS.) $D(n) = \sum_{i=1}^{16} D(N_i) + O(n^4)$, where $N_1, \ldots, N_{16} \leq n/2$.

2 (100 PTS.) The secretary problem.

The company WeSnoozeYouLose[2] (WSYL) specializes in online hiring of workers. They had developed their revolutionary online platform that enables them to interview up to t candidates for a job in a single ***round***. The system then rank these t candidates in an decreasing order of their quality. Thus, the output of a round is a list $C_1, C_2, \ldots, C_t$, where C_1 is the best candidate, etc.

WSYL was hired to figure out who are the best k candidates out of n given candidates, For simplicity, assume that $t > 2$ is odd and $k = \lceil t/2 \rceil$). Because of EU law, WSYL can not transfer test results from one round to the next, and must retest all the candidates invited to a round from scratch (but they are allowed to invite a candidate to multiple rounds). Describe an algorithm performing, as few rounds as possible, that identify the best k candidates.

Provide an upper bound on the number of rounds your algorithm performs (naturally, the smaller and tighter the upper bound the better), as a function of n and t (we usually do not care about constants, but here we do – provide an explicit upper bound, without using O notation). For example, how many rounds does your algorithm performs for $n = 49$ and $t = 7$ (here $k = 4$)?

[2] Valued at 57 billion dollars in 2019, currently about to go bankrupt.

Comments:

(A) We assume a total order on the candidates – there are no ties.
(B) WSYL can use information from rounds already completed, such as the ranking inside each round, to figure out the top k candidates.
(C) Getting the right answer here is somewhat hard. Any reasonable answer would get full credit, but for your own benefit, you might want to figure out what the right answer is (roughly).

3 (100 PTS.) Random sorting.

The input is an array $A[1 \,..\, n]$ of n real numbers. Consider the randomized algorithm, that in each iteration scans the array and check if it is sorted in increasing order. If it is sorted, then it stops. Otherwise, it randomly picks two indices (uniformly and independently) $i, j \in [\![n]\!] = \{1, \ldots, n\}$. If $i < j$ and $A[i] > A[j]$ then it swaps the two locations. You might want to convince yourself that sooner or later this algorithm would stop with probability 1. Prove an upper bound (as small as you can) on the expected number of iterations till this algorithm stops. (One can figure out asymptotically the exact number of expected iterations, but this is quite difficult. Any proof of a polynomial bound in n is worth full credit here.)