

Due February 6, 2:45pm

Instructions: Write your name, the username for your instructional account, your TA's name, and your discussion section time prominently on the first page of your homework. Also list your study partners for this homework, or "none" if you had no partners.

Any time that you are asked for an algorithm, please provide: (a) the pseudocode of your algorithm, (b) a brief explanation of the main idea/intuition underpinning your algorithm, (c) a proof of correctness, (d) the asymptotic running time of the algorithm, using $\Theta(\cdot)$ notation, (e) justification for your running time calculation.

1. (20 pts.) Divide and conquer

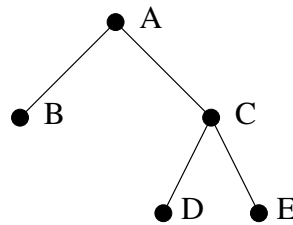
The base commander at the local military base has decided to set up a phone tree that he can use to pass along a message to every soldier stationed at the base, in case of emergency. Assume that every soldier at the base has exactly one commanding officer at the base (except the base commander). Assume that each officer knows the phone numbers for all of his/her direct subordinates. In other words, we will assume that the military hierarchy forms a tree rooted at the base commander. In this problem, each officer can have arbitrarily many subordinates.

The base commander has asked you to devise an algorithm for computing an optimal schedule for passing the message down the tree. We will assume that as soon as an officer receives the message, the officer immediately begins calling his/her direct subordinates, one at a time. Your schedule should specify, for each officer, the order in which the officer should contact his/her subordinates. Assume that each phone call takes exactly one minute, and that no one has call waiting or three-way calling, so that each person can be on the phone with at most one other person at any time. The latency of a schedule is the time it takes from when the base commander starts the process until the last soldier receives the message and hangs up. Your algorithm should produce a schedule with the lowest possible latency and output the latency of this schedule. The running time of your algorithm should be at most $O(n \lg n)$, where n denotes the number of soldiers.

You should: provide pseudocode for your algorithm; concisely explain the main idea/intuition behind your algorithm; justify why it is correct (i.e., why the schedule it outputs will eventually end up notifying every soldier, and why the schedule it produces has the smallest possible latency); state the running time of your algorithm as a function of n using $\Theta(\cdot)$ notation; and justify your running time analysis.

(continued on next page)

For instance, suppose there are $n = 5$ soldiers and the tree looks like this:



Here are two possible schedules for that tree:

Schedule 1	Schedule 2
A: B, C	A: C, B
C: D, E	C: D, E

(For instance, in Schedule 1, the base commander A calls B first, then calls C as soon as he has hung up on B. When C is called, he first calls D, then calls E.) Note that the latency of schedule 1 is 4 minutes, while the latency of schedule 2 is 3 minutes (since the phone call from C to D can proceed at the same time as the phone call from A to B). Therefore, schedule 2 is better.

2. (25 pts.) Practice with recurrence relations

Solve the following recurrence relations. Express your answer using $\Theta(\cdot)$ notation. (For instance, if you were given the recurrence relation $T(n) = T(n-1) + 3$, the solution would be $T(n) \in \Theta(n)$.) You don't need to show your work or justify your answer for this problem.

- (a) $F(n) = F(n-3) + 1$.
- (b) $G(n) = G(n/2) + 3$.
- (c) $H(n) = H(n/4) + 1$.
- (d) $I(n) = I(n/2) + n$.
- (e) $J(n) = 2J(n/4) + 5$.
- (f) $K(n) = 2K(n/4) + n$.
- (g) $L(n) = 2L(3n/4) + 1$.
- (h) Let $M(n) = M(n/4) + M(3n/4) + 1$. Find an asymptotic upper bound on $M(n)$, i.e., an explicit function $f(n)$ such that $M(n) \in O(f(n))$. I do not care about constant factors. Prove that your upper bound is a valid upper bound. Try to find the best asymptotic upper bound you can, but don't spend too long on this question—you do not need to find the asymptotically best possible upper bound. You do not need to find a matching lower bound, and you do not need to prove that $M(n) \in \Theta(f(n))$.

Optional challenge problem: For extra credit, solve the recurrence: i.e., find $f(n)$ such that $M(n) \in \Theta(f(n))$, and prove your answer. Be warned that this may be challenging.

Revised 2/3: Part (h) was revised to ask only for an upper bound, and not necessarily the best possible bound.

3. (15 pts.) Practice with polynomials and complex numbers

The purpose of this problem is to familiarize you with computation using polynomials and complex numbers, to help you follow the lectures on the fast Fourier transform. In the following, let $\omega = e^{2\pi i/8} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i$.

- (a) Calculate $\omega + \omega^7$, to 2 significant digits.
- (b) Let $p(x) = x^2 + 1$. Evaluate $p(1)$, $p(\omega)$, $p(\omega^2)$, and $p(\omega^3)$. Simplify your answer as much as possible.
- (c) Find the unique polynomial $q(x)$ of degree at most 3 (with complex coefficients) that satisfies $q(1) = 3$, $q(\omega) = 1 + \sqrt{2}i$, $q(\omega^2) = 1$, and $q(\omega^3) = 1 + \sqrt{2}i$. Feel free to do this computation either by hand or using software; if you use software, be sure to show the setup and explain what you did.

4. (20 pts.) Out of sorts

Consider the following sorting algorithm:

WeirdSort($A[0..n-1]$):

1. If $n = 1$, return.
2. If $n = 2$:
3. If $A[0] > A[1]$, swap $A[0]$ and $A[1]$.
4. If $n \geq 3$:
5. Let $k := \lceil 2n/3 \rceil$.
6. Use WeirdSort to sort the first k items in A . [i.e., call $\text{WeirdSort}(A[0..k-1])$]
7. Use WeirdSort to sort the last k items in A . [i.e., call $\text{WeirdSort}(A[n-k..n-1])$]
8. Use WeirdSort to sort the first k items in A . [i.e., call $\text{WeirdSort}(A[0..k-1])$]

It turns out that this algorithm will correctly sort the input array. Let's analyze its running time.

- (a) Let $T(n)$ = the number of comparisons between array elements when executing WeirdSort on an array of size n . Write a recurrence relation for $T(n)$.
- (b) Solve the recurrence relation you wrote down in part (a).
Hint: You should be able to manipulate your answer into the form $T(n) \in \Theta(n^c)$, for some constant c . What value of c did you get?
- (c) Based on your answer to part (b), would you expect WeirdSort to be faster than, slower than, or about the same speed as insertion sort?

5. (20 pts.) The brain-scrambler

Previous problems too easy? OK, try this one! Design an efficient algorithm that accepts an array $A[0..n-1]$ of n integers as input and finds indices ℓ, r that minimize the sum $\sum_{i=\ell}^{r-1} A[i]$, subject to the constraint that $0 \leq \ell \leq r \leq n$.

Try to come up with an algorithm that is as fast as possible, ignoring constant factors. List the pseudocode for your algorithm; explain the main idea; justify why it is correct; state its asymptotic running time using $\Theta(\cdot)$ notation; and justify your running time analysis.

Amazingly, it is possible to solve this in $O(n)$ time. But if you can't find an $O(n)$ time solution, try to find the fastest algorithm you can. Doing it in $O(n^3)$ time is trivial. $O(n^2)$ time is not too hard. But can you do better than $O(n^2)$?

Warning: Some array elements of $A[]$ may be negative; some may be positive. Notation: If $r = \ell$, then $\sum_{i=\ell}^{r-1} A[i] = 0$. You may assume that each addition operation takes $O(1)$ time.