CS 170 Algorithms Fall 2014 David Wagner HW 2

Due Sept. 12, 6:00pm

Instructions. This homework is due Friday, September 12, at 6:00pm electronically. It must be submitted electronically via Pandagrader (not in person, in the drop box, by email, or any other method). We recommend you submit your homework by 5:00pm, as the Pandagrader website sometimes becomes overloaded and slow in the final hour before the deadline.

You are welcome to form small groups (up to four people) to work through the homework, but you **must** write up all your solutions strictly by yourself, and you must acknowledge any ideas you got from others (including from books, papers, web pages, etc.) Please read the collaboration policy on the course web page.

On the first page of your submission, put your name, your student ID number, your class account userid, the homework number (HW2), and your study partners for this homework or "none" if you had no partners. Note that you will need a class account, and you will need to log into it and complete the registration process.

Each problem should begin on a new page. The pages of your homework submissions must be in order (all pages of problem 1 in order followed by all pages of problem 2 in order, etc...). See the web site for detailed instructions on how to prepare your PDF file and upload to Pandagrader. Make sure to select pages in Pandagrader for each question. You risk receiving no credit for any homework that does not adhere to these guidelines.

No late homeworks will be accepted. No exceptions. Please don't ask for extensions. We don't mean to be harsh, but we prefer to make model solutions available shortly after the due date, which makes it impossible to accept late homeworks.

1. (10 pts.) Big-Theta running time

In class, we saw big-O notation: e.g., $5n^2 = O(n^3)$. It is also useful to know $\Theta(\cdot)$ notation, so this question will give you practice with that. Roughly speaking, big-O notation is like an asymptotic version of $\leq$, where we don't care about constants. Intuitively, $\Theta(\cdot)$ notation is like an asymptotic version of $=$, where we don't care about constants. More precisely, we write $f(n) = \Theta(g(n))$ if (a) $f(n) = O(g(n))$ and (b) $g(n) = O(f(n))$. (See Chapter 0.3 of the textbook for more details.)

With that in mind, answer the following questions with Yes or No. You do not need to provide any explanation or justification.

(a) Is $5n^2 = \Theta(n^3)$?

(b) Is $5n^2 = \Theta(n^2)$?

(c) Is $5n^2 + 7n + 1 = \Theta(n^2)$?

(d) Is $n \lg n = \Theta(n^2)$?

2. (16 pts.) Practice with running time analysis

Consider the following two algorithms:

Algorithm F(n):
1. For $i := 0, 1, \ldots, n-1$:
2. (do something)

Algorithm G(n):
1. For $i := 0, 1, \ldots, \lceil \lg n \rceil - 1$:
2. (do something)

(a) Suppose that step 2 takes i^2 steps of computation in the ith iteration. What is the total running time of algorithm F, as a function of n, using $\Theta(\cdot)$ notation? Show your calculation.

(b) Suppose that step 2 takes $n - 2i$ steps of computation in the ith iteration (except that when $2i \geq n$, it takes 1 step of computation). What is the total running time of algorithm F, as a function of n, using $\Theta(\cdot)$ notation? Show your calculation.

Revised 9/7 to clarify what happens if $2i \geq n$.

(c) Suppose that step 2 takes $n/2^i$ steps of computation in the ith iteration. What is the total running time of algorithm G, as a function of n, using $\Theta(\cdot)$ notation? Show your calculation.

(d) Suppose that step 2 takes $n/(i+1)$ steps of computation. Prove that the total running time of algorithm F is $O(n \log n)$.

Hint: Look up the harmonic series online. Or, upper-bound each term $n/(i+1)$ by $n/2^k$ for some appropriately chosen k (possibly different for each term).

Revised 9/7 to avoid dividing by zero.

3. (15 pts.) Out of sorts

Consider the following sorting algorithm:

Algorithm $S(A[0..n-1])$:
1. If $n = 2$ and $A[0] > A[1]$, swap $A[0]$ and $A[1]$.
2. If $n \geq 3$:
3. Let $k := \lceil 2n/3 \rceil$.
4. Call $S(A[0..k-1])$. ("Sort the first two-thirds.")
5. Call $S(A[n-k..n-1])$. ("Sort the last two-thirds.")
6. Call $S(A[0..k-1])$. ("Sort the first two-thirds.")

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 S on an array of size n. Write a recurrence relation for $T(n)$.

(b) Solve the recurrence relation you wrote down in part (a). Express your solution using $\Theta(\cdot)$ notation.

Hint: You should be able to write your answer in the form $T(n) = \Theta(n^c)$, for some constant c.

(c) Based on your answer to part (b), would you expect S to be faster than, slower than, or about the same speed as insertion sort?

4. (20 pts.) Merge asymptotics

After grading the exams, the CS 170 staff want to sort all of the exams according to the student ID numbers, so that they can easily retrieve one if the need arises. There are n exams and k GSIs. Each GSI gathers n/k

exams and sorts them into a pile. But then the GSIs leave and now David wants to merge all of these piles together. Merging two piles, one with a exams and the other with b exams, takes $\Theta(a+b)$ work. David has two options:

1. Merge pile 1 with pile 2, then merge the result with pile 3, and then merge the result with pile 4, and so on.
2. Split the piles into two roughly equal halves, recursively merge each half, and then use the merge procedure to combine the two halves.

(a) How much work does David do if he uses method 1? Write a recurrence relation, then solve it. Express your final answer using $\Theta(\cdot)$ notation.

(b) How much work does David do if he uses method 2? Write a recurrence relation, then solve it. Express your final answer using $\Theta(\cdot)$ notation.

(c) Which method is better?

5. (19 pts.) Plurality finding: divide-and-conquer

Definition. An array $A[0..n-1]$ is said to have a $1/3$-plurality element if some value v appears $> n/3$ times in the array; each such value v is called a $1/3$-plurality element.

Design a divide-and-conquer algorithm that, given $A[0..n-1]$, outputs "Yes" and a $1/3$-plurality element if $A[0..n-1]$ has a $1/3$-plurality element, or "No" if $A[0..n-1]$ does not have a $1/3$-plurality element. (If $A[0..n-1]$ has multiple $1/3$-plurality elements, the algorithm can return any one of them.) Your algorithm should have $O(n \lg n)$ running time.

However, there is a special restriction. The only thing you are allowed to do with array elements is compare whether they are identical (test whether $A[i] = A[j]$ for some i, j of your choice). The array elements are not from an ordered domain, so you cannot compare them using $<$ and $>$, and you cannot hash the array elements.

If you like, you may assume that $n = 2^k \times 3$ for some k, to avoid dealing with annoying edge cases.

Revised 9/7 to allow the assumption that n is three times a power of two.

Note: in this class, whenever we ask you to design an algorithm on a homework set, your write-up should include all of the following parts:

- *Explanation:* Explain the main idea behind your algorithm. Conciseness is good: try to explain in at most a few sentences. A good goal is that if another CS 170 student were to read this part, they'd say "oh! now I see how to solve this problem"—you should be explaining the key insight that lets them solve the problem. Don't just repeat what the pseudocode says.
- *Algorithm:* Give the pseudocode of your algorithm.
- *Running time:* State the running time of your algorithm. Then, justify this claim—show the calculation.
- *Proof of correctness:* Prove that your algorithm is correct, i.e., always produces a correct result. State and prove any invariants that help demonstrate its correctness.

Label each of these main parts in your answer, to make it easy for readers to find them.

6. (20 pts.) More divide-and-conquer

You are given a list of n intervals $[x_i, y_i]$, where x_i, y_i are integers with $x_i \le y_i$. The interval $[x_i, y_i]$ represents the set of integers between x_i and y_i. For instance, the interval $[3,6]$ represents the set $\{3,4,5,6\}$. Define the

overlap of two intervals I, I' to be $|I \cap I'|$, i.e., the cardinality of their intersection (the number of integers that are included in both intervals).

Devise a divide-and-conquer algorithm that, when given n intervals, finds and outputs the pair of intervals with the highest overlap. (You can resolve ties arbitrarily.)

It's easy to find an algorithm whose running time is $\Theta(n^2)$. Look for something better.

Hint: try splitting using the left endpoint of the intervals.

As always, include all of the major parts listed above (explanation, algorithm, running time, proof of correctness) in your answer.