CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, FEBRUARY 18, 2022 AT 11:30 PM VIA GRADESCOPE **A2BASIC(WYZ5RGE)**

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: spj6s, jhs8cue

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *Asymptotics*

1. Write a mathematical statement using the appropriate order-class to express "Algorithm A's worst-case $W(n)$ is quadratic."

   Solution: $W(n) \in O(n^2)$

2. Write a mathematical statement using the appropriate order-class to express "Algorithm A's time-complexity $T(n)$ is never worse than cubic for any input."

   Solution: $T(n) \in O(n^3)$

3. Write a statement using words and an appropriate order-class to express "It's not possible for an algorithm that solves problem P to succeed unless it does at least a cubic number of operations."

   Solution: $P(n) \in \Omega(n^3)$

4. Prove or disprove the following statement: $n(\log n)^2 \in O\left(n^{1.5}(\log n)\right)$.

   Solution:

   Proof. Let $c = 2$, such that for sufficiently large n, $n \log n^2 \leq cn^{1.5}(\log n)$
   Then, for $n = 1$,

$$\begin{aligned} n \log n^2 &= (1) * (\log(1))^2 \\ &= 0 \end{aligned}$$

$$\begin{aligned} cn^{1.5} \log n) &= 2 * ((1)^{1.5}(\log 1) \\ &= 0 \end{aligned}$$

   Thus, $\forall n > 1, n \log n^2 \leq 2n^{1.5} \log n$, by direct proof $\square$

PROBLEM 2 *Basic Sorting*

1. In a few sentences, explain if changing the comparison done in mergesort's *merge()* function from $\leq$ to $<$ makes the sorting algorithm incorrect, and also whether it makes the sort unstable.

   Solution: It would remain correct because using an inclusive inequality ($\leq$) instead of exclusive ($<$) would result in the swapping of elements that have the same value, which would not affect the correctness of the sort. It would, however, be an unstable sort, as the sort could be in different orders, and thus the algorithm is not necessarily deterministic

2. Which of the following are true about insertion sort and mergesort?

   (a) Insertion sort would run reasonably fast when the list is nearly in reverse-sorted order but with a few items out of order.

   (b) For small inputs we would still expect mergesort to run more quickly that insertion sort.

   (c) The lower-bounds argument that showed that sorts like insertion sort must be $\Omega(n^2)$ does not apply to mergesort because when a list item is moved in *merge()* it may un-do more than one inversion.

   (d) We say the cost of "dividing" in mergesort is 1 because we must do a constant amount of work to find the midpoint of the subproblem we're sorting.

   Solution: (c) and (d) are true

PROBLEM 3 *Recurrence Relations*

1. Reduce the following recurrence to its closed form (i.e. remove the recursive part of its definition) using the *unrolling method.*

$$T(n) = 3T(n/3) + n \text{ and } T(1) = 1$$

   Be sure to show the general form of the recurrence in terms of how many times you've "unrolled", as well as a formula for how many times you "un-roll" before getting to the base case.

   Solution:

$$\begin{aligned} T(n) &= 3[3T(n/9) + n/3] + n \\ &= 9T(n/9) + 2n \\ &= 9[3T(n/27) + n/9] + 2n \\ &= 27T(n/27) + 3n \\ &\ldots \end{aligned}$$

   We repeat this for i repetitions so that $n/3^i$ tends towards 1 and $i = log_3 n$ and we arrive at a general form

$$\begin{aligned} T(n) &= 3^i T(n/3^i) + in \\ &= 3^{\log_3 n} T(n/3^{\log_3 n}) + (\log n)n \\ &= n \log n \end{aligned}$$

2. Use the Master Theorem to find the order-class for this recurrence: $T(n) = 3T(n/2) + n \log n$. State which case applies, and if no case applies and the Master Theorem cannot be used, state that and explain why.

   Solution:
   For $T(n) = 3T(n/2) + n \log n$,
   $a = 3$, $b = 2$, and $f(n) = n \log n$
   This is Case 1 of the Master Theorem:

   $$f(n) \in O(n^{\log_b a - \epsilon}) \Longrightarrow \log n \in O(n^{\log_2 3})$$

   Thus $T(n) \in \Theta(n^{log_2 3})$

3. Use the Master Theorem to find the order-class for this recurrence: $T(n) = 3T(n/4) + n \log n$. State which case applies, and if no case applies and the Master Theorem cannot be used, state that and explain why.

   Solution:
   For $T(n) = 3T(n/4) + n \log n$,
   $a = 3$, $b = 4$, and $f(n) = n \log n$
   This is Case 3 of the Master Theorem:

   $$f(n) \in \Omega(n^{\log_b a + \epsilon}) \Longrightarrow \log n \in \Omega(n^{\log_4 3})$$

   Thus $T(n) \in \Theta(n \log n)$

4. Show you understand how to do a proof using the "guess and check" method and induction. Show that the following recurrence $\in O(n \log_2 n)$:

   $$T(n) = 4T(n/4) + n \text{ and } T(1) = 1$$

   You can assume n is a power of 4.

   Hints: For the induction, you have to prove the relationship for a small value of n. You'll find $n = 1$ doesn't work, but you can show it holds for the next larger value of n. (Again, assume n is a power of 4.) It's OK for the induction proof if the relationship holds for some small value of n even if it doesn't hold for $n = 1$.
   Also, you'll need to guess a value for c. For this problem, the value of c is not anything strange or unusual. A small value will work, you will find it easiest to just keep c in your math calculations and when you get to the final step you can see what value of c makes your relationship true. (This problem is much easier than the example we did in class!)

   Solution:

   Proof. Let $f(n) = cn \log_2 n$, $c \in N$
   We proceed by induction.
   Base Case: $n = 4$

   $$\begin{aligned} T(4) &= 4T(4/4) + 4 \\ &= 4T(1) + 4 \\ &= 8 \end{aligned}$$

   $$\begin{aligned} f(4) &= 4c \log_2 4 \\ &= 8c \end{aligned}$$

Thus for all $c \in N$, $T(n) \leq f(n)$ holds for base case $n = 4$ and $T(n) \in O(nlog_2 n)$

Inductive hypothesis: For sufficiently large n, $T(n) \leq cn \log_2 n$ Inductive step: $n = n + 1$

$$T(n+1) = 4T(\frac{n+1}{4}) + (n+1)$$

$$f(n+1) = c(n+1)\log_2(n+1)$$

Because $\frac{n+1}{4} \leq$n,T(n + 1) $\leq f(n+1)$. Because our inductive hypothesis holds for our base case and for all cases onwards, it must be the case that $T(n) \in O(n \log n)$ □

PROBLEM 4 *Divide and Conquer #1*

Write pseudo-code that implements a divide and conquer algorithm for the following problem. Given a list L of size n, find values of the largest and second largest items in the list. (Assume that L contains unique values.)

In your pseudo-code, you can indicate that a pair of values is returned by a function using Python-like syntax, if you wish. For example, a function $funky()$ that had this return statement:

```
return a, b
```

would could be used to assign a to x and b to y if called this way:

```
(x, y) = funky()
```

Solution:

```
divide(list, a, b, max1, max2)
    If b - a <= 1   # Base case
        return max(list.at(a), list.(b))
    max1 = divide(list, a, (a+b)/2, max1, max2)
    max2 = divide(list, (a+b)/2 + 1, b, max1, max2)
    return max(max1, max2)  # Recursive call

main()
    # Call divide() on initial list, where a is the first index and b is the list index,
    # and max1 & max2 parameters are the values at the indices a & b
    Max1 = divide(list, a, b, list.at(a), list.at(b))

    # Remove the max element from the array
     list.remove(Max1)

    # Call divide() on array again, now with max removed, to find second largest value
    Max2 = divide(list, a, b, list.at(a), list.at(b))
```

PROBLEM 5 *Divide and Conquer #2*

Conference Superstar. There is a CS conference with n attendees. One attendee is a "superstar" — she is new to the field and has written the top paper at the conference. She is the attendee whom all other attendees know, yet she knows no other attendee. Specifically, if attendee a_i is the superstar, then $\forall a_j \neq a_i$, $knows(a_j, a_i) == true$ and $knows(a_i, a_j) == false$. Other attendees may or may not know each other, as is true for "normal" meetings. Give a $O(n)$ algorithm which determines who the superstar is.

Hint: Compare pairs of attendees and try to eliminate one of them. Then you might want to do a swap for each comparison to make sure all attendees that have a certain property are together in one part of your list so you can recurse on just those.

Solution: Our strategy here will involve maintain two lists, one for a list of attendees, and one for a list of attendees who we we have determined to not be the superstar. We will divide the list of attendees into pairs and compare each pair with $knows(a, b)$. Every time $knows(a, b)$ returns *true*, then attendee a knows attendee b and thus cannot be the superstar, so we remove them from the list of attendees and move them to the second list of ruled-out attendees. We keep doing this until there is only one person left in the main list. This one person is remaining because she is the only person who doesn't know anyone else, thus satisfying our condition for the superstar.

PROBLEM 6 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added. You should only submit your `.pdf` and `.tex` files.