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:

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: *For Algorithm A, $W(n) \in \theta(n^2)$*

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: $\forall n, T(n) \in O(n^3)$

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: *For an algorithm A that solves Problem P, $A \in \Omega(n^3)$.*

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

   Solution:

$$n(\log n)^2 \le n^{1.5}(\log n)$$
$$n \log n \le n^{1.5}$$
$$\log n \le n^{0.5}$$

   For $n_0 = 1$, c = 1:

$$\log n_0 \le n_0^{0.5} * c$$
$$\log 1 \le 1^{0.5} * 1$$
$$0 \le 1$$

   Therefore, $n(\log n)^2 \in O\left(n^{1.5}(\log n)\right)$.

PROBLEM 2 *Basic Sorting*

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

Solution: *Changing the function does not make the sorting algorithm incorrect because the comparison still correctly compares the values of two elements. However, it does make the algorithm unstable.*

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), (d)*

PROBLEM 3 *Recurrence Relations*

1. Reduce the following recurrence to its closed form (i.e. remove the recursive part of its defininiton) 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:
   $T(n/3) = 3T((n/3)/3) + n/3 = 3T(n/9) + n/3$
   $T(n/9) = 3T((n/9)/3) + n/9 = 3T(n/27) + n/9$
   $T(n) = 3(3T(n/9)+n/3) + n = 9T(n/9) + 2n$
   $\quad\quad = 9(3T(n/27)+n/9) + 2n = 27T(n/27) + 3n$
   Pattern: $T(n) = 3^i T(n/3^i) + in$
   $T(n/3^i) = T(1) = n/3^i = 1 \rightarrow i = log_3 n$
   $T(n) = 3^{log_3 n} T(n/3^{log_3 n}) + nlog_3 n = nT(1) + nlog_3 n = n + nlog_3 n$

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: *$a = 3$ and $b = 2$, so $log_b a = log_2 3 = 1.58 > 1$, so this is Case 1. Therefore, the answer is $\theta(n^{log_2 3})$.*

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: *$a = 3$ and $b = 4$, so $log_b a = log_4 3 = 0.79 < 1$. Also, $af(n/b) \le cf(n)$ for $c = 0.9$. This means it is Case 3. Therefore, the answer is $\theta(nlogn)$.*

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:

Goal: $T(n) \leq 16 n \log_2 n = O(n \log_2 n)$

Base Case: $T(4) = 4T(4/4) + 4 = 4T(1) + 4 = 8 \leq 4 \log_2 4$
$\qquad\qquad T(16) = 4(16/4) + 16 = 4T(4) + 16 = 48 \leq 16 \log_2 16$

Hypothesis: $\forall n \leq x_0, T(n) \leq 16 n \log_2 n$

$$
\begin{aligned}
T(x_0 + 1) &= 4T(\tfrac{x_0+1}{4}) + x_0 + 1 \\
&= 4(16(\tfrac{x_0+1}{4}) \log_2(\tfrac{x_0+1}{4})) + x_0 + 1 \\
&= 16(x_0 + 1) \log_2(\tfrac{x_0+1}{4}) + x_0 + 1 \\
&= 16(x_0 + 1)(log_2(x_0 + 1) - log_2 4) + x_0 + 1 \\
&= 16(x_0 + 1)(log_2(x_0 + 1) - 2) + x_0 + 1 \\
&= 16(x_0 + 1)(log_2(x_0 + 1) - 32(x_0 + 1) + x_0 + 1 \\
&= 16(x_0 + 1)(log_2(x_0 + 1) - 32x_0 - 32 + x_0 + 1 \\
&= 16(x_0 + 1)(log_2(x_0 + 1) - 31x_0 - 31 \leq 16(x_0 + 1)(log_2(x_0 + 1)
\end{aligned}
$$

Therefore, $T(n) \in O(n \log_2 n)$

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:

```
L1 = [beginning:middle]
L2 = [middle:end]
x1, x2, x3, x4 = 0
for i in L1:
      if i > x1, x1 = i
      if i > x2 and i < x1, x2 = i
for i in L2:
      if i > x3, x3 = i
      if i > x4 and i < x3, x4 = i
if x1 > x3
      if x2 > x3, return x1, x2
      else, return x1, x3
if x3 > x1
      if x4 > x1, return x3, x4
      else, return x3, x1
```

"""""

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:

1. Put all the attendees in pairs (a_1 and a_2) and perform the following action for each pair.

   (a) If $knows(a_1, a_2) == true$, add a_1 to the end of the list and a_2 to the beginning of the list.

   (b) If $knows(a_1, a_2) == false$, add a_1 to the beginning of the list and a_2 to the end of the list.

2. The list of possible superstars will be in the beginning half of the list. Recursively call the algorithm on the part of the list with the possible superstars until there is only one person in that part of the list left. That person is 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.