Six questions; each question is worth 20 points. There are 120 points for graduate students and 100 points for undergraduates.

1. Time complexity upper and lower bounds are defined as follows:

   (a) **Time complexity upper bound** is a time threshold in which all instances of a problem can be solved.

   (b) **Time complexity lower bound** is a time threshold below which some instances of the problem cannot be solved.

   (c) These notions apply to both problems and algorithms.

Now assume algorithm A solves problem Π. Also assume that \( U(n) \) and \( L(n) \) are a time upper bound and a time lower bound for problem Π, respectively. Let \( t_A(n) \) and \( s_A(n) \) be the time upper bound and the time lower bound for algorithm A, respectively. Are the following relationships necessarily correct? Please justify for your answers.

   (a) \( U(n) \leq t_A(n) \) ?

   (b) \( U(n) \geq s_A(n) \) ?

   (c) \( L(n) \geq t_A(n) \) ?

   (d) \( L(n) \leq s_A(n) \) ?

2. Using the definition of Big-O to prove or disprove

   (a) If \( T(n) = O(f(n)) \) and \( S(n) = O(f(n)) \), \( T(n) + S(n) = O(f(n)) \).

   (b) If \( T(n) = O(f(n)) \), then \( T(n)f(n) \neq O(f(n)) \).
3. (Graduate students only, bonus for undergraduates) Consider the following summation of the harmonic sequence up to the \( n \)th term, for any \( n \geq 1 \),

\[
h(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}
\]

(1) Prove that we have the following recurrence for \( h(n) \). (No math induction is needed in your proof.)

\[
h(n) \leq h(\lfloor n/2 \rfloor) + 1
\]

(2) Use the substitution method to prove that, there are constant \( c > 0, k > 0 \) such that

\[
h(n) \leq c \log_2 n
\]

for all \( n \geq k \).

4. Find an upper bound (the big-\( O \) notation) for the following recurrence:

\[
T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + \frac{n}{2}, \quad T(1) = 1
\]

Use the substitution method (i.e., guess and check) to prove your upper bound.

5. Let function \( T(n) = 4T(\frac{n}{2}) + n \), where \( T(1) = 1 \). Use the recursive tree method to derive an upper bound for \( T(n) \).

6. The Insertion Sort algorithm discussed in the class (also in the textbook) can be written as recursive algorithms, by changing either or both of the loops in Insertion Sort to recursive calls. This exercise asks you to rewrite Insertion Sort to a recursive algorithm, namely Rec-Insertion Sort, by replacing only the outer for loop with a recursive call.

You would need to use the following recursion scheme for the recursive insertion sort. Let \((A, n)\) be an array with \( n \) elements to be sorted. The algorithm first sorts the prefix \( n - 1 \) items and then inserts the last item \( A[n] \) into the sorted prefix.

(1) Give such a recursive algorithm for insertion sort.

(2) Someone has suggested that the steps for the inner while loop could be done more efficiently since the prefix sublist has already been sorted. For example, one could use a “binary search” to quickly identify where the new item can be inserted, which could be done in \( O(\log_2 n) \) time for each new item. So the total time would be \( O(n \log_2 n) \) instead of \( O(n^2) \). Show that the time complexity would actually remain the same as \( O(n^2) \).
The answers must be word-processed or typed. You may substitute formulae and figures with hand-writings. Your submitted algorithms should be in the pseudo-code, not in any specific programming language. Answers deviating from these requirements will be returned without grading.

The answers must be the student’s own work. Idea sharing and referencing to others’ work (including those online) are not allowed. Plagiarism and other forms of academic dishonesty will be handled within the guidelines of the Student Handbook and reported to the University.