COMPUTER SCIENCE 311
SPRING 2020
MIDTERM EXAM 2
APRIL 15 8:00 AM TO APRIL 17, 7:59 AM

INSTRUCTIONS

- You have **48** hours to complete the exam, including the uploading time of your solutions.
- There are four questions and one bonus question. The maximum possible score is 70; the number of points for each problem is indicated on the next page.
- Read all questions carefully before starting.
- Since the exam is conducted remotely, questions should be asked on **Piazza only**. Note that we do **NOT** answer questions through emails.
- All questions should be only for clarification purposes. Please keep your questions concise. We can NOT have lengthy discussions like we usually answer homework questions on Piazza.
- Due to the challenges of conducting the exam remotely, we can only try to answer the questions in time when we see them. However, we cannot guarantee how fast we can respond. If you have questions, please ask early.
- We expect clear and concise answers. Think carefully before you write; make sure that every word you write counts.
- If you do not know the answer to a problem, write "*I don't know*" and you will earn approximately 20% of the points for that problem.
- When asked to provide an algorithm, remember to
    - describe the algorithm clearly,
    - analyze the running time of the algorithm.
- For any algorithm, your grade will depend on the efficiency of your algorithm.

WRITE YOUR NAME AND YOUR RECITATION SECTION NUMBER HERE

YOUR SCORE

Problem	Points	Score
1	15	
2	15	
3	15	
4	15	
Bonus	10	
Total	70	

(a) (7 points) Short questions; no justifications needed, except for partial credit. No credit for writing "I don't know."

   (i) (2 points) Consider the following recurrence: $T(n) = 4T(n/2) + n^2$, $T(1) = 1$. What is $T(8)$?

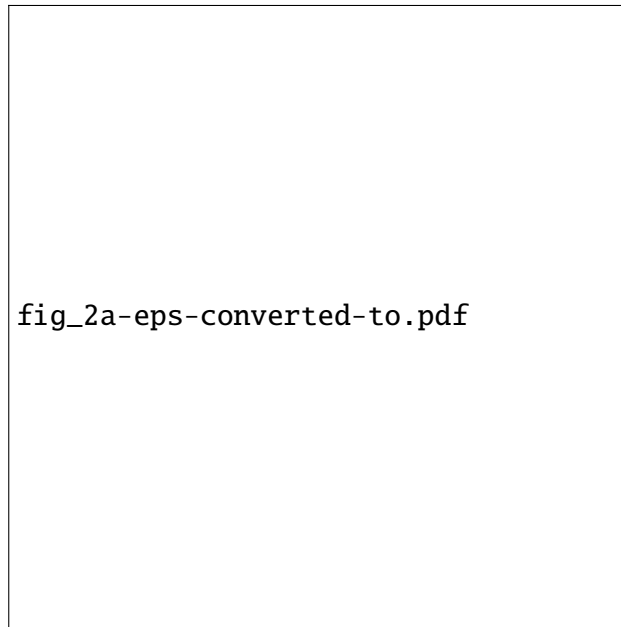   (ii) (2 points) Which case of the Master Theorem applies, 1, 2, or 3?

   (iii) (3 points) What is the asymptotic bound of $T(n)$?

(b) (5 points) Solve the following recurrence: $T(n) = 2T(n/4) + T(n/2) + n$, $T(1) = 1$. You must show how you arrive at the solution.

(a) (7 points) Consider the undirected graph G below.

fig_2a-eps-converted-to.pdf

(i) (3.5 points) Draw the breadth-first search tree for G assuming that the search starts at node a and that ties among nodes are broken according to the alphabetical order.

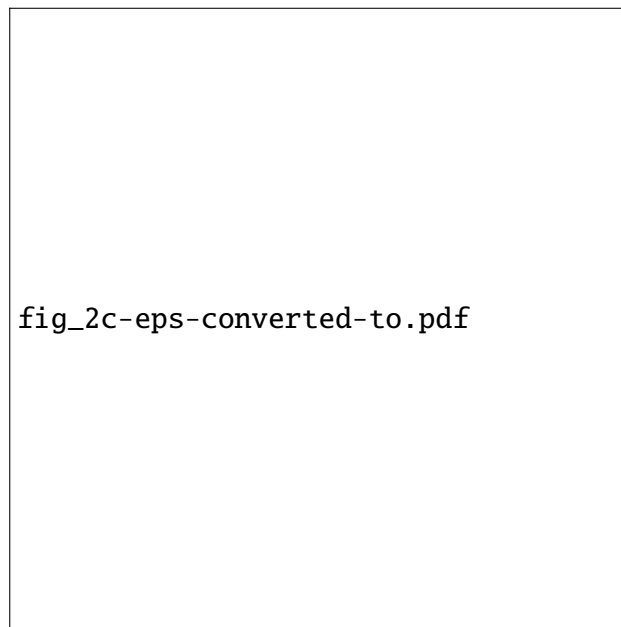(ii) (3.5 points) Draw the depth-first search tree for G assuming that the search starts at node a and that ties among nodes are broken according to the alphabetical order.

(b) (4 points) Draw two topological orderings of the following graph.
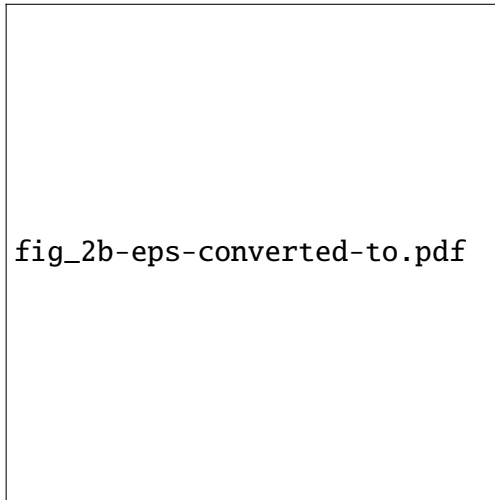
(c) (4 points) State all strongly connected components of the following graph (in the form of $\{a, b, ...\}, \{...\}, ...$).

(a) (7 points) Let $G = (V, E)$ be a directed graph. The length of each edge in G is one. Define $G^2 = (V, E')$ as follows: for vertices $u, v \in V$, $(u, v) \in E'$ if there is a path of length two between u and v in G. Suppose that a directed graph is given as adjacency matrix. Design an algorithm to compute G^2. Derive the running time of your algorithm.

(b) (8 points) Given a directed graph $G = (V, E)$ with $|V| = m$ and $|E| = n$, a vertex v is said to be a ground vertex if there is a path from every other vertex of the graph to v. Note that a graph may have multiple ground vertices. Given the adjacency list representation of G, design an algorithm to find a ground vertex if one exists. If the graph has multiple ground vertices, then it suffices to output one of them. Derive the running time of your algorithm. Your grade will depend on the efficiency of your algorithm.

Suppose that $\texttt{Subroutine}(A, B)$ takes as argument two arrays of integers A, B and returns an array. The running time is $O(n)$, where n is the number of elements of A plus the number of elements in B. The above algorithm is used in the recursive $\texttt{Mystery}$ algorithm given below.

```
 1 Mystery(A)
 2     if n == 1 then
 3         return A[0]
 4     A₁ = ⟨A[0]...A[n/2 − 1]⟩              // The first half of A.
 5     A₂ = ⟨A[n/2]...A[n − 1]⟩              // The second half of A.
 6     A₃ = ⟨A[n/4]...A[3n/4 − 1]⟩           // The middle half of A.
 7     B₁ = Mystery(A₁)
 8     B₂ = Mystery(A₂)
 9     B₃ = Mystery(A₃)
10     C₁ = Subroutine(B₁, B₂)
11     C₂ = Subroutine(B₂, B₃)
12     C₃ = ⟨C₁[0]...C₁[n/2 − 1], C₂[n/2]...C₂[n − 1]⟩   /* Concatenating 1st half
        of C₁ and 2nd half of C₂ */
13     return C₃                             // Constant O(1) time.
```

Let $T(n)$ denote the running time of Mystery on an array of n elements.

(a) (7 points) Write the recurrence equation for $T(n)$. (Hint: You may use the following facts: (1) the formula for the geometric series $\sum\limits_{k=0}^{m-1} ar^k = a\left(\frac{r^m - 1}{r - 1}\right)$, for any $r \neq 1$; and (2) $a^{\log_b n} = n^{\log_b a}$.)

(b) (8 points) What is the running time of `Mystery` on n-element array? Justify your answer. You are not allowed to use the Master Theorem.

Suppose that A is an array of natural numbers, i.e., elements in $\{1, 2, 3, \ldots\}$. An element x in A is called a **majority element** if *strictly* more than half the array cells contain value x. For example, 1 is the majority element of $[1, 2, 1, 1]$. Suppose that the following algorithm is available to us:

`IsMajElement` (B, x): Takes in an array B of natural numbers and a natural number x and returns true if and only if x is the majority element in B. The running time is $O(n)$, where n is the length of B.

(a) (4 points) Complete the following *recursive* algorithm which takes in an array A of natural numbers, and returns the majority element x if there is one, and -1 otherwise. Note: No points will be given for non-recursive solutions. (*Hint:* In order for some element x to be a majority element in an array, it must be a majority element of either the left half, the right half, or both.)

```
1  findMajority(A)
2      if (|A| == 1) then
3          return A[0]
       // You pseudocode here:
```

(b) (3 points) What is the recurrence equation of your algorithm? (No justification required, just write the recurrence).

(c) (3 points) What is the running time of your algorithm? (No justification required, just write the running time in Big-O notation)

SCRATCH PAPER

SCRATCH PAPER

SCRATCH PAPER