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

Benjamin Purdy 730639934

Your Score

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

Problem 1: Divide-and-Conquer (15 Points)

(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 $T(8)$?
threw the recurence method we get logn+1 $\{n^2 + ... + n^2\}$
hence $n^2 logn$
With a depth of 7 the recurrence is $64 \times 7 = 448 = \frac{64}{448} = \frac{8}{54}$

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

Solution. The master Theorem shows us that $T(n) = 4T(n/2) + n^2$ a = 4 , b = 2, f(n) = n^2.
This gives us $log_2 4 = 2$
Now since f(n) = $log_b a$ this is case 2 giving us a running time for $\Theta(n^{log_b a} logn) = \Theta(n^2 logn)$

□

(iii) (3 points) What is the asymptotic bound of $T(n)$?
$T(n) = \Theta(n^{log_b a} logn) = \Theta(n^2 logn)$

(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.

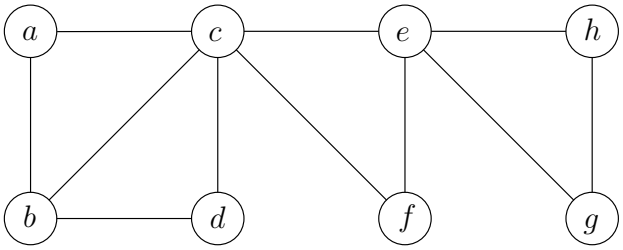
Solution. Guess:T(n)≤cnlogn, for some c. then replace T($\frac{n}{2}$) with $c(\frac{n}{2})(log_2(\frac{n}{2})$ and T($\frac{n}{4}$) with$c(\frac{n}{4})(log_2(\frac{n}{4})$

$$
\begin{aligned}
T(n) &= 2T(n/4) + T(n/2) + n \\
&\leq 2c \cdot \frac{n}{4} \cdot log_2 \frac{n}{4} + c\frac{n}{2} \cdot log_2 \frac{n}{2} + n \\
&= c \cdot \frac{n}{2}\left(log\frac{n}{4} + log\frac{n}{2}\right) + n \\
&= c \cdot \frac{n}{2}\left(logn - log4 + logn - log2\right) + n \\
&= c \cdot \frac{n}{2} \cdot logn\left(- log4 - log2\right) + n \\
&= c \cdot \frac{n}{2} \cdot logn\left(- log(\frac{4}{2})\right) + n \\
&= c \cdot \frac{n}{2} \cdot logn - 1 + n \\
&= \frac{c}{2} nlogn + n - 1 \\
&=\leq cnlogn + n
\end{aligned}
$$

Now as you can see cnlogn +n is O(nlogn) □

Problem 2: Graphs (15 Points)

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



(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.

Solution. I've done BFS adding all children starting from the left and moving up.
So In a stack starting at a $a = \begin{bmatrix} b \\ c \end{bmatrix}$ poping b gives us a stack $ab = \begin{bmatrix} c \\ d \end{bmatrix}$

$$now\ abc = \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

$$abcd = \begin{bmatrix} e \\ f \end{bmatrix}$$

$$abcde = \begin{bmatrix} f \\ g \\ h \end{bmatrix}$$

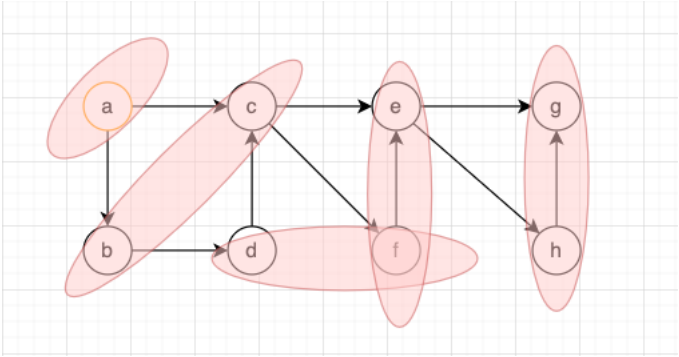
$$abcdef = \begin{bmatrix} g \\ h \end{bmatrix}$$

$$abcdefg$$

□

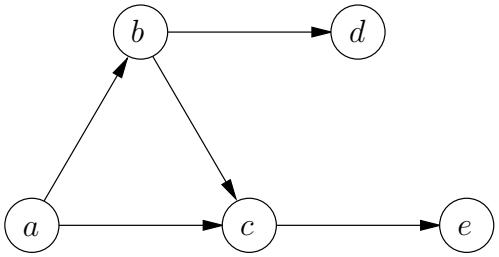
(ii) (3.5 points) Draw the depth-first search tree for G assuming that the search starts at node a and that ties among nodes are broken according to the alphabetical order.

Solution. .
there are 4 depths in this graph. d,f,e are one depth sorry for the picture.



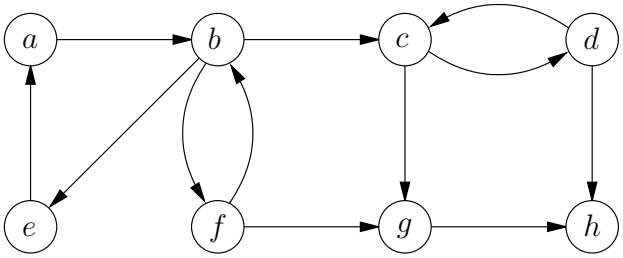
□

(b) (4 points) Draw two topological orderings of the following graph.



A. $\begin{bmatrix} a & b & c & d & e \end{bmatrix}$
B. $\begin{bmatrix} a & b & d & c & e \end{bmatrix}$

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



$[a, b, e], [f, b], [c, d]$
$[g], [h], [a, b, e, f]$

Problem 3: Graphs (15 Points)

(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.

```
groundvertices
    if V = 1x1 array then
        return empty 1x1 matrix;
    // do matrix multiplication ;
    Graph c;
    for i = 0 to V[G] do
        for j = 0 to V[G] do
            for k = 0 to V[G] do
                c[i, j] = c[i, j] + c[i, k] · c[k, j];
    return c
```

Proof. Nested loops from 0 → V length means that the time complexity $V \cdot V \cdot V = O(V^3)$

□

(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.

```
1  groundvertices
2      if Size of V = 1 then
3          return V[0];
4      List = size of V;
5      for i : V[i] do
6          DFS(V,V[i]);
7          for i : list[i] do
8              if V[i] = visited then
9                  V[i] = unvisited;
10                 list[i]++;
11     for i : list[i] do
12         if list[i] = V.length then
13             return i;
14     return -1
```

```
1  DFS( V , i )
2      for i : threw all adj edges do
3          if visited[i] == false then
4              mark node as visited;
5              DFS(V,adj.edge) ;
6      return current index;
```

Proof. Efficiency of your algorithm is baced on DFS which we know is O(V+E). V number of calls to DFS are done to astablish a list of size v that shows how much vertexs have access to that node. as the end if V DFS calls I search the list for a number of v connections. doing so is time O(V) this gives us a running time for $O(V(V + E) + V) = O(V^2 + VE)$ now we know that E is equal to V(V-1) meaning that $O(V^2 + VE) = O(V^2 + V(V(V - 1))) = O(V^2 + V^3 - V^2) = O(V^3)$ oof □

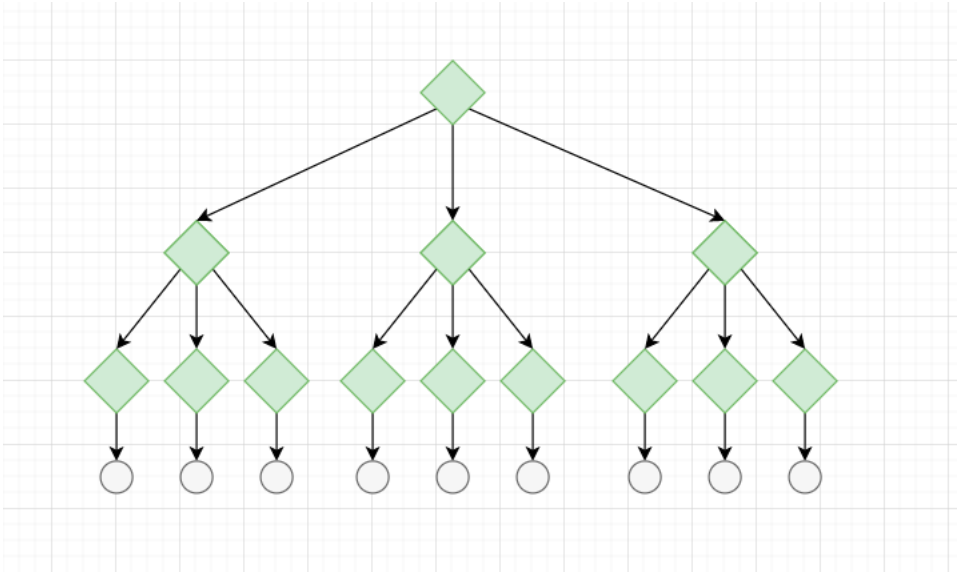
Problem 4: Dvide and Conquer (15 Points)

Suppose that `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 `Mystery` algorithm given below.

```
1  Mystery(A)
2      if n == 1 then
3          return A[0]
4      A_1 = ⟨A[0] ... A[n/2 − 1]⟩                          // The first half of A.
5      A_2 = ⟨A[n/2] ... A[n − 1]⟩                          // The second half of A.
6      A_3 = ⟨A[n/4] ... A[3n/4 − 1]⟩                       // The middle half of A.
7      B_1 = Mystery(A_1)
8      B_2 = Mystery(A_2)
9      B_3 = Mystery(A_3)
10     C_1 = Subroutine(B_1, B_2)
11     C_2 = Subroutine(B_2, B_3)
12     C_3 = ⟨C_1[0] ... C_1[n/2 − 1], C_2[n/2] ... C_2[n − 1]⟩   /* Concatenating 1st half
           of C_1 and 2nd half of C_2 */
13     return C_3                                           // 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 $\sum_{k=0}^{m-1} ar^k = a\left(\frac{r^m-1}{r-1}\right)$, for any $r \neq 1$; and (2) $a^{\log_b n} = n^{\log_b a}$.)



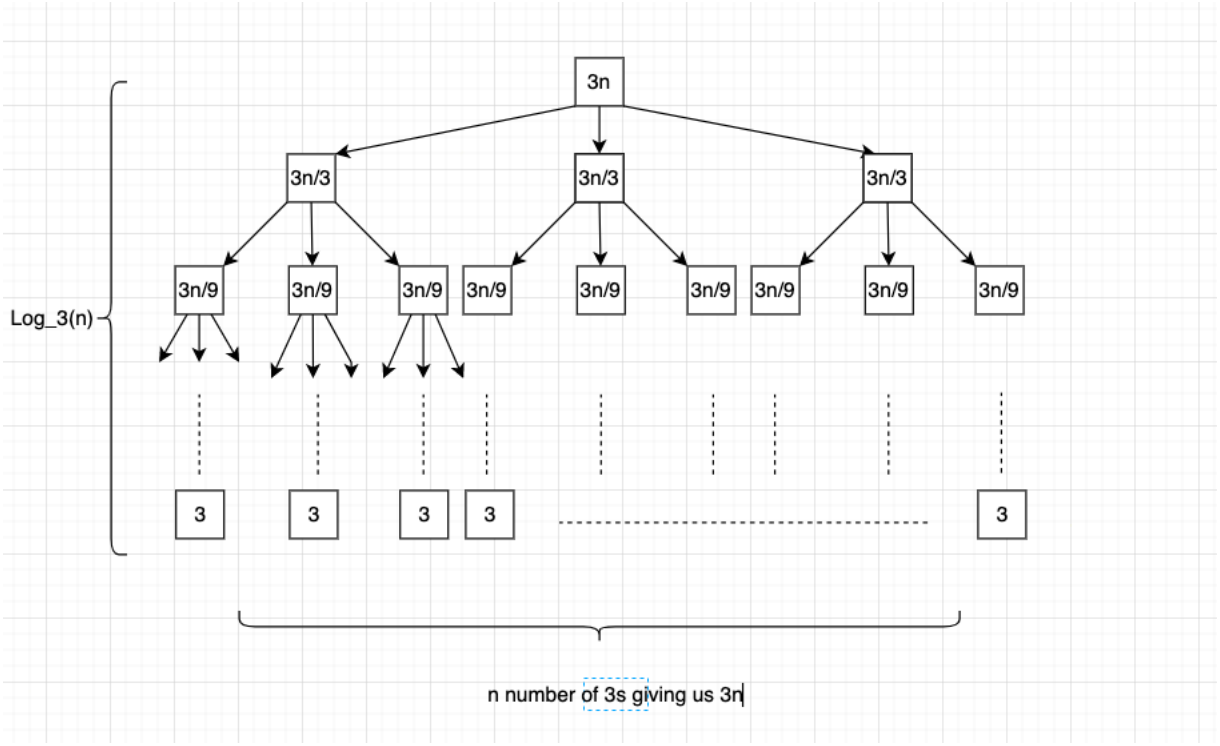
Solution.

by following the algoritm we can see that the base case is T(1) = 1
T(2) = a split of 3 nodes off of origonal = 3+1 = 4 T(3) = 3 splits of T(2) giving us 9 children of T(2) = 3+9+1 =13 Now in this recursive function T(n/3) will give the recurence.

This gives us the start to our equation T(n) = aT(n/3) + b by spliting the array into 3 parts this gives us a = 3 nowC_1, C_2, C_3 all run in n times giving us T(n) = 3T(n/3)+3n □

(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.

image. .



Proof. By recursion we can see that the total is $3nlog_3 n + 3n$ this in turn is $O(nlogn)$

□

The Substitution Method
Recurrence:T(n) =3T(n/3) +3n, with T(1) = 1.
Guess:T(n)$\leq c(n/3)log_3(n/3)$, for some constant c.
Consider $n > 1$

$$\begin{aligned} T(n) &= 2T(n/2) + 3n \\ &\leq 2(c(n/3)log_3(n/3)) + 3n \\ &\leq 2cnlog_3 n - 2nlog_3 3 + 3n = 2cnlog_3 n - 2n + 3n \\ &= 2cnlog_3 n + n \end{aligned}$$

thus T(n) =$O(nlogn)$.

□

Problem 5: Bonus Question: (10 Points)

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]
4      size = n/2
5      x_1 = findMajority([A[0], ··· , A[n/2]]
6      x_2 = findMajority([A[n/2] + 1, ··· , A[n]]
7      if x_1 = x_2 then
8          return x_1
9      SubMajority_1 = isMajElement(A, x_1)
10     SubMajority_2 = isMajElement(A, x_2)
11     if SubMajority_1 then
12         return x_1
13     if SubMajority_2 then
14         return x_2
15     return -1
```

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

$T(n) = 2T(n/2) + n$

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

$O(nlogn)$

Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper