CSE 100: Algorithm Design and Analysis
Final Exam

Summer 2020

- This is a take-home exam with no proctoring. You can start at any time once you get access to this exam. You have to submit your solution by **11:59pm, 16-AUG**, through CatCourses (Final Exam under Assignments). You can resubmit any number of times until the deadline. If there are any technical issues with uploading, it is extremely important to immediately contact the instructor or the TA.
- This is an open-book exam. The **only** resources you can use during the exam are all **course materials** uploaded to CatCourses plus the **textbook**. In particular, this means that you are NOT allowed to search for solutions on the internet. No calculator is allowed. You have to take the exam by yourself.
- There are 13 problems in total. You can earn some partial points if you show progress even if you can't solve problems completely.
- Please make your answers concise as well as precise. If there is something in the question that you believe is open to interpretation, then please go ahead and interpret, but state your assumptions in your answer.
- If you have questions, you can email the instructor and the TAs. However, we may not answer your questions on time as we can't be available all the time. In particular, do not expect reply on Monday.

You're required to take the following honor pledge prior to taking the exam.

> *By completing this exam, I acknowledge and confirm that I will not give or receive any unauthorized assistance on this examination. I will conduct myself within the guidelines of the university academic integrity guidelines.*

1. (15 points). If the statement is correct, choose 'true,' otherwise 'false.' Each subproblem is worth 1 point.

   (a) $n^2 = \Omega(n \log n)$.

   (b) $2^n = O(n^{100} \log n)$.

   (c) $\sum_{i=1}^{n} i = O(n)$.

   (d) The decision tree of any comparison based sorting algorithm has a height $\Omega(n \log n)$.

   (e) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.

   (f) If $T(n) = 2T(n/2) + n$, then we have $T(n) = O(n \log n)$.

   (g) If $T(n) = T(\frac{9}{10}n) + n$, then we have $T(n) = O(n)$.

   (h) The adjacency matrix representation of a graph $G = (V, E)$ needs memory $O(|E|+|V|)$.

   (i) If G is undirected, then G and G^T have the same adjacency matrix representation.

   (j) There is a linear time algorithm that finds a median in $O(n)$ time.

   (k) If an undirected graph $G = (V, E)$ has $|V| - 1$ edges, it must be a tree.

   (l) If an undirected graph $G = (V, E)$ is connected and its edges have distinct weights, G has a unique minimum spanning tree.

   (m) One can build a max-heap in $O(n)$ time.

   (n) In the (s-t) max flow problem, the maximum flow value is equal to the minimum cut value.

   (o) If $T(n) = T(n/2) + \Theta(n)$, then we have $T(n) = \Theta(n \log n)$.

2. (6 points). Just give the algorithm name(s):

   (a) (2 points) Name two sorting algorithms whose (worst-case) running time is $O(n \log n)$.

   (b) (2 points) Given a directed graph $G = (V, E)$ where all edges have a unit weight/distance, and two vertices $s, t \in V$, we want to find a shortest path from s to t in $O(E + V)$ time. Which algorithm would you use?

   (c) (2 points) Given a directed graph $G = (V, E)$ where every edge has positive weight/distance, and two vertices $s, t \in V$, we want to find a shortest path from s to t. Which algorithm would you use? You should use the fastest algorithm.

3. (5 points) The following is a pseudocode of the naive divide-and-conquer algorithm for matrix multiplication. Here, partitioning a n by n matrix means partitioning it into four $n/2$ by $n/2$ (sub-)matrices.

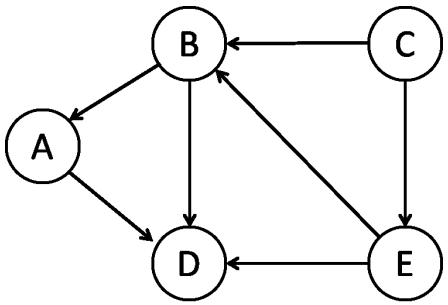
```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
 1  n = A.rows
 2  let C be a new n × n matrix
 3  if n == 1
 4      c11 = a11 · b11
 5  else partition A, B, and C as in equations (4.9)
 6      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
 7      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
 8      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
 9      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10  return C
```

Give the recurrence for the running time of Square-Matrix-Multiply-Recursive and solve it. Your solution should be formatted as shown below. We let $T(n)$ denote the running time when input matrices are n by n. No need to show how you solved it. Just state the final result along with the recurrence.

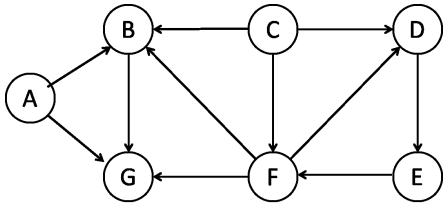
Recurrence:
$T(n) =$

After solving the recurrence,
$T(n) =$

4. (5 points) Find a topological sort of the following graph using DFS. You must *label* each vertex with its DFS finishing time and *list* the vertices according to the resulting topological sort. (Please conside vertices in increasing alphabetical order. Also visit each vertex's neighbors in increasing alphabetical order.)



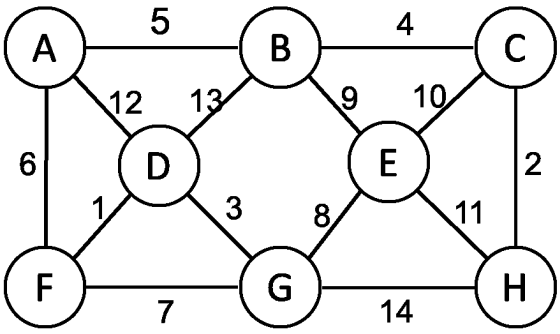
5. (10 points) Consider the following directed graph.



(a) (5 points) Do depth-first search in the graph G (NOT the vertex G), considering vertices in increasing alphabetical order (i.e. start with vertice A). Also visit each vertex's neighbors in increasing alphabetical order. Show the final result, with vertices labeled with their starting (discovery) and finishing times. The DFS forest, or DFF, must be clear.

(b) (5 points) Based on your results, proceed to run the algorithm we learned in class to find the strongly connected components of G (show the result of the DFS with vertices labeled with their starting (discovery) and finishing times. The DFF must be clear.)

6. (6 points) Consider the following weighted undirected graph.



(a) (3 points) We would like to run the Kruskal's algorithm on this graph. List the edges appearing in the Minimum Spanning Tree (MST) in the order they are added to the MST.

(b) (3 points) We would like to run the Prim's algorithm on this graph using A as initial vertex. List the edges appearing in the Minimum Spanning Tree (MST) in the order they are added to the MST.

7. (6 points) Suppose we have a disjoint-set data structure that supports the following operations with their respective running time.

- Make-Set(x): $O(1)$ time.
- Union(x, y): $O(\log n)$ time, where n is the number of all elements considered.
- Find-Set(x): $O(1)$ time.

Suppose we implement the following pseudo-code of Kruskal's algorithm using this data structure. We want to analyze the running time. Answer the following three questions. Your answer must be an asymptotic quantity in terms of $|V|$ and/or $|E|$.

MST-KRUSKAL(G, w)
1. $A = \emptyset$.
(E.g., the running time of Line 1 is $O(1)$)
2. for each vertex $v \in G.V$,
3. Make-Set(v)

(1 points). What is the running time of Lines 2 and 3?

4. Sort the edges of $G.E$ into non-decreasing order by weight w (using the best sorting algorithm)
(2 points). What is the running time of Line 4?
5. for each edge $(u, v) \in G.E$, taken in non-decreasing order by weight
6. if Find-Set$(u) \neq$ Find-Set(v)
7. $A = A \cup \{(u, v)\}$
8. Union(u, v)
(3 points). What is the running time of Lines 5-8?
9. Return A.

8. (7 points) For your convenience, we provide the following pseudocodes:

Initialize-Single-Source(G, s)
1. for each vertex $v \in G.V$
2. $v.d = \infty$
3. $v.\pi = NIL$
4. $s.d = 0$

Relax(u, v, w)
1. if $v.d > u.d + w(u, v)$
2. $v.d = u.d + w(u, v)$
3. $v.\pi = u$

The following is an incomplete pseudo-code of Dijkstra's algorithm. Complete it. If you're not comfortable, you can give a pseudocode from scratch. Recall that G is an input graph and s is the source vertex. You may find the following notation useful: $G.Adj[u]$ - u's neighbor vertices.

Dijkstra(G, w, s)
1. Initialize-Single-Source(G, s)
2. $S = \emptyset$ // The set of vertices whose shortest distance has been finalized.
3. $Q = G.V$ // Add all vertices to min priority queue Q.

9. (4 points) The Dijkstra's algorithm may fail to find a shortest path if the graph has negative weight edges – in other words, after running the Dijkstra algorithm, we have $v.d \neq \delta(s, v)$ for some $v \in V$. Give such a graph that *has no negative-weight cycle and has no more than four edges*. Note that you must specify the source vertex and the edge weights.

10. (5 points) Give a pseudocode of the Search operation of binary search tree. Recall that the input to the Search operation is a pointer to a node x and a key k. The operation should output a pointer to a node with key x in the subtree rooted at x if such a node exists, and return NIL otherwise. You can use either iteration or recursion. Please use the following notation: $x.key$ is x's key; $x.left$ points to x's left child; and $x.right$ points to x's right child. Name your procedure *Tree-Search* with inputs x and k.

11. (6 points) Quicksort implementation. Give a *pseudo-code* of Quicksort(A, p, r) that sorts $A[p...r]$ via quick-sort. You can assume that all elements in A have distinct values. You can use the following helper function, Partition(A, p, r):

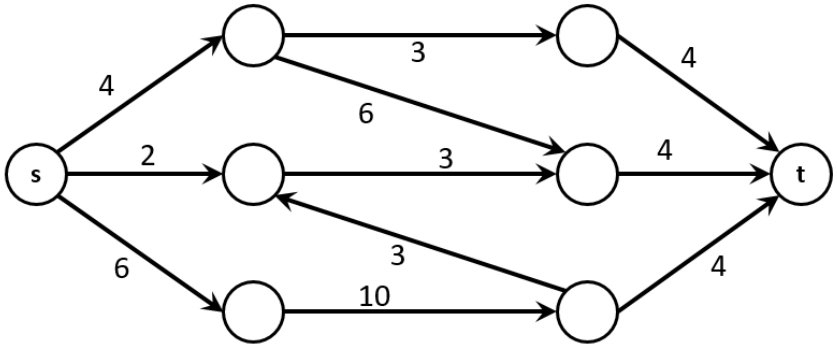
```
Partition(A, p, r)
```

```
1. x = A[r]
2. i =  p - 1
3. for j = p to r-1
4.    if A[j] <= x
5.       i = i+1
6.       exchange A[i] with A[j]
7. exchange A[i+1] with A[r]
8. return i +1
```

12. (15 points) Do the following:

    (a) (5 points) Write the pseudo-code of Ford-Fulkerson. Specifically include the pseudo-code of the function increaseFlow(path P in G_f, flow f). It is ok to use BFS to find a path from s to t in the residual network.

    (b) (10 points) For the following graph, show step-by-step the how Ford-Fulkerson finds the max flow/min cut between s and t. At each iteration, show the graph of the flow and the residual network. (Hint: there should be four (4) iterations to find the optimal solution)



13. (10 points) The following is a pseudo-code of Bellman-Ford. Prove its correctness. More specifically, you must prove that if there is no negative-weight cycle reachable from s, then the algorithm ensures that $v.d = \delta(s, v)$ at the end and returns TRUE; otherwise, it returns FALSE.

    Bellman-Ford(G, w, s)
    1. Initialize-Single-Source(G, s)
    2. for $i = 1$ to $|G.V| - 1$
    3. for each edge $(u, v) \in G.E$
    4. Relax(u, v, w)
    5. for each edge $(u, v) \in G.E$,
    6. if $v.d > u.d + w(u, v)$
    7. return FALSE
    8. return TRUE