CSE 100: Algorithm Design and Analysis
Final Exam

Spring 2019

Note: Please write down your name on every page. The total max score of all problems, except the last one, is exactly 100. Any points above 100 will be considered as extra points.

Problem	Points earned	Max
1		24
2		6
3		6
4		4
5		12
6		8
7		6
8		8
9		4
10		10
11		6
12		6
13		20
Sum		Max 100

Name _____

1. (24 points). If the statement is correct, choose 'true,' otherwise 'false.' Each subproblem is worth 2 points.

   (a) $n^2 = \Omega(n \log n)$.
   True False **Sol.** true

   (b) $2^n = O(n^{100} \log n)$.
   True False **Sol.** false

   (c) $\sum_{i=1}^{n} i = O(n)$.
   True False **Sol.** false

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

   (e) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.
   True False **Sol.** true

   (f) If $T(n) = 2T(n/2) + n$, then we have $T(n) = O(n \log n)$.
   True False **Sol.** true

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

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

   (i) If G is undirected, then G and G^T have the same adjacency matrix representation.
   True False **Sol.** true

   (j) There is a linear time algorithm that finds a median in $O(n)$ time.
   True False **Sol.** true

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

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

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

   (a) (2 points) Name two sorting algorithms whose (worst-case) running time is $O(n \log n)$. **Sol.** Merge, Heap. 1pt for each name

   (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? **Sol.** BFS

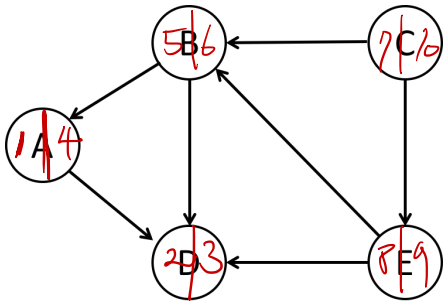
   (c) (2 points) Given a directed graph $G = (V, E)$ where every edge has positive weight/distance, and two vertices $s, t \in V$, we want to find a shortest path from s to t. Which algorithm would you use? You should use the fastest algorithm. **Sol.** Dikjstra

3. (6 points). Suppose we just ran DFS on a directed graph G. The algorithm DFS we learned in class computes $(v.d, v.f)$ for every vertex $v \in V$. Using this, we would like to do the following. Briefly explain how. If your solution is not based on each vertex's discovery/finish time, you cannot get full points.

   (a) (3 points) How do we check if an edge (u, v) is a back edge? **Sol.** If u's interval, which is defined by u's discovery and finish times, is a sub-interval of v's interval, then (u, v) is a back edge; otherwise, not.

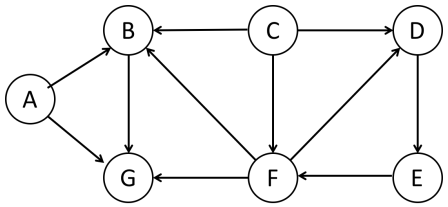
   (b) (3 points) How do we compute the number of depth first trees? **Sol.** Among intervals $(v.d, v.f), v \in V$, count the number of intervals that are not contained by any other intervals.

4. (4 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.)

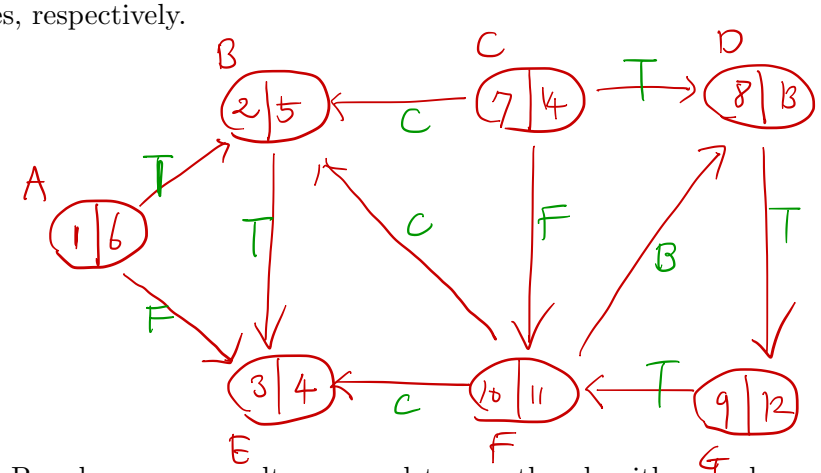


C, E, B, A, D

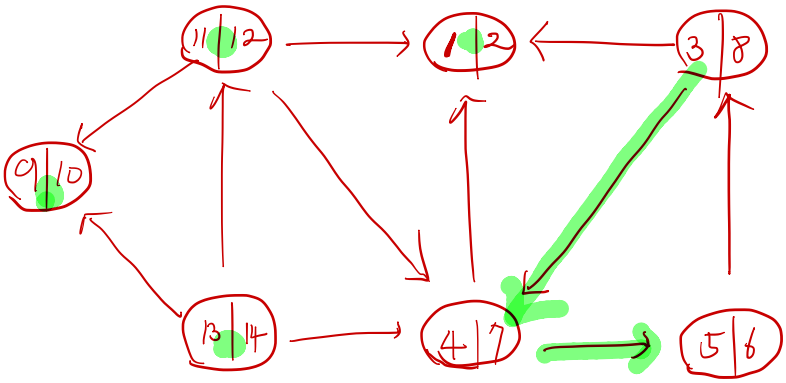
5. (12 points) Consider the following directed graph.



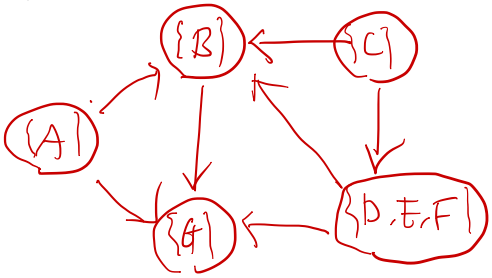
(a) (6 points) Do depth-first search in G, considering vertices in increasing alphabetical order. Also visit each vertex's neighbors in increasing alphabetical order. Show the final result, with vertices labeled with their starting (discovery) and finishing times, and edges labeled with their type (T/B/F/C); here, T, B, F, C stand for tree, back, forward and cross edges, respectively.



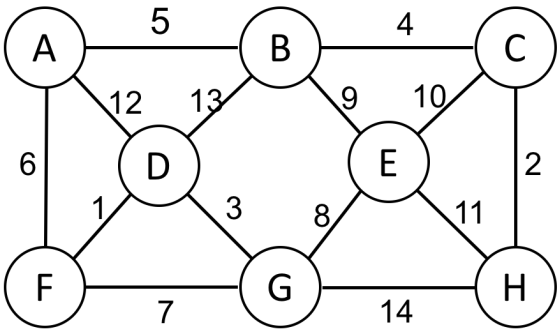
(b) (3 points) Based on your results, proceed to run the algorithm we learned in class to find the strongly connected components of G (show the result of the DFS in G^T, with vertices labeled with their starting (discovery) and finishing times. The DFFs must be clear.)



(c) (3 points) Draw the component graph G^{SCC} of G.



6. (8 points) Consider the following weighted undirected graph.



(a) (2 points) Explain why edge (A, F) is safe. In other words, give a cut where the edge is the cheapest edge crossing the cut. **Sol.** A possible solution: $\{D, E, F, G\}$, and the others.

(b) (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. **Sol.** DF, CH, DG, BC, AB, AF, GE

(c) (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. **Sol.** AB, BC, CH, AF, DF, DG, GE

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? Sol. $O(V)$.

   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? Sol. $O(E \log E)$ or $O(E \log V)$.

   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? Sol. $O(E \log V)$.

   9. Return A.

8. (8 points) For your convenience, we provide the following pseudocodes:
   Initialize-Single-Source(G, s)
   1. for each vertex $v \in G.V$
   2. $\quad v.d = \infty$
   3. $\quad v.\pi = NIL$
   4. $s.d = 0$

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

   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

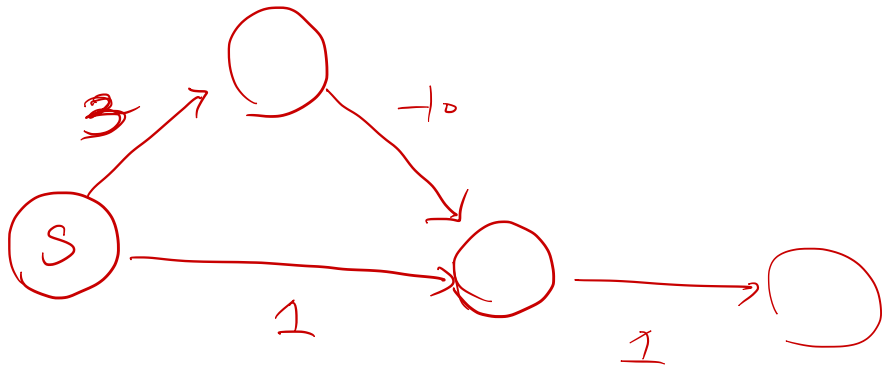
   Dijkstra(G, w, s)
   1. Initialize-Single-Source(G, s)
   2. $S = \emptyset$ // The set of vertices whose shortest distance has been finalized.
   3. $Q = G.V$ // Add all vertices to min priority queue Q.

   Sol. See the lecture notes/CLRS.

dist[s] ←0
for all v ∈ V–{s}
do dist[v] ←∞ S←∅

&bgqr_lac rm qmspac tcprcv gq xcpm'
&qcr _jj mrfcp bgqr_lacq rm glglgrw'
&Q* rfc qcr md tgqgrcb tcprgacq gq glgrg_jjw cknrw' &O* rfc oscsc glgrg_jjw amlr_glq _jj tcprgacq' &u fgjc rfc oscsc gq lmr cknrw'
&qcjcar rfc cjckclr md O ugrf rfc kgl, bgqr_lac' &_bb s rm jgqr md tgqgrcb tcprgacq'
bm
u ← mindistance(Q,dist) S←S∪{u}
for all v ∈ neighbors[u]
do if dist[v] > dist[u] + w(u, v) (if new shortest path found)
then d[v] ←d[u] + w(u, v) (set new value of shortest path) (if desired, add traceback code)
return dist

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.



≠ a possible sol.

10. (10 points)

(a) 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: $x.key$: x's key; $x.left$ points to x's left child; and $x.right$ points to x's right child.
Tree-Search$(x; k)$ **Sol.** See the lecture notes/CLRS.

(b) Give a pseudocode of Inorder-Tree-Walk(x), which outputs nodes in a BST rooted at x according to inorder.
Inorder-Tree-Walk(x)

Sol. See the lecture notes/CLRS.

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
```

Sol. See the lecture notes/CLRS.

12. (6 points) In the rod cutting problem, we are given as input, $p_0, p_1, p_2, ..., p_n$, where p_i denotes the price of a rod/piece of length i. We are interested in cutting a given rod of length n into pieces of integer lengths in order to maximize the revenue; here we are only interested in finding the maximum revenue. Cutting is free. Let r_i denote the maximum revenue one can get out of a rod of length i.

    - (3 points) Give a recursion for computing r_i, where $i = 0, 1, 2, ..., n$. Do not forget the base case(s). **Sol.** $r_i = \max\{p_1 + r_{i-1}, p_2 + r_{i-2}, \cdots, p_{i-1} + r_1, p_i\}$ when $i \geq 1$; and $r_0 = 0$. One can choose the base case to be when $i = 1$.

    - (3 points) Fill out the following DP table. Please double-check your calculation, as there will be no partial points for this problem. Each entry is worth 1pt.

i	1	2	3	4	5	6
p_i	1	3	4	5	6	7
r_i	1	3	4			

    Sol. 6, 7, 9. (-1 pt for each incorrect sol.)

13. (20 points) (Note: This is a bonus problem. This problem is for students who challenged themselves hard to learn more. This problem is not for collecting partial points. So, unless your answer is formal enough, you will likely get 0 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

    Sol. See the lecture notes/CLRS.