Midterm One

Name:

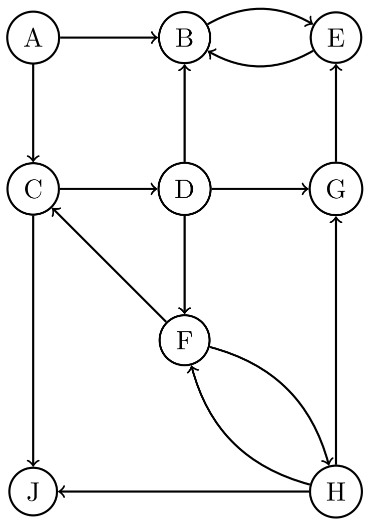
SID:

GSI and section time:

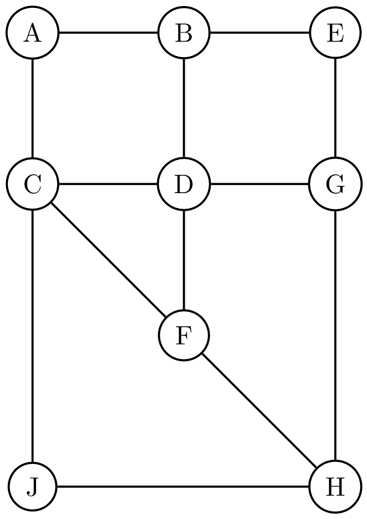
Answer all questions. Read them carefully first. Be precise and concise. The number of points indicate the amount of time (in minutes) each problem is worth spending. Not all parts of a problem are weighted equally. Write in the space provided, and use the back of the page for scratch. Box numerical final answers. Good luck!

When an explanation is required, answer with one or two short sentences. *(20 points)*

1. For the directed graph below, find the strongly connected components and draw the DAG of strongly connected components.

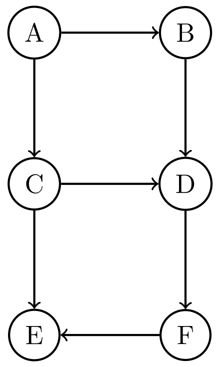


2. Execute DFS on the following undirected graph starting at node D breaking ties alphabetically. Mark the pre and post values of the nodes.



Node	pre	post
A		
B		
C		
D		
E		
F		
G		
H		
J		

3. In an implementation of Bellman-Ford, starting with the initialization $dist(A) = 0, dist(B) = dist(C) = dist(D) = dist(E) = dist(F) = \infty$, the following sequence of updates are applied on the graph shown below.



1) $update(A \to C)$
2) $update(B \to D)$
3) $update(A \to B)$
4) $update(C \to D)$
5) $update(F \to E)$
6) $update(C \to E)$
7) $update(D \to F)$
8) $update(B \to D)$
9) $update(D \to F)$
10) $update(F \to E)$
11) $update(C \to D)$
12) $update(A \to C)$
13) $update(A \to B)$
14) $update(C \to E)$
15) $update(D \to F)$
16) $update(B \to D)$
17) $update(A \to B)$
18) $update(C \to D)$
19) $update(F \to E)$
20) $update(C \to E)$

What is the earliest step at which the distance to F guaranteed to be correct, for all possible weights on the edges? Justify your answer.

4. Describe the naive algorithm for Fourier transform. What is its running time?
(Briefly and precisely describe the algorithm, no need to prove the correctness)

`Input:` $a_0, \ldots, a_{n-1} \in \mathbb{R}$
`Output:` `the Fourier transform` $b_0, \ldots, b_{n-1}$ `using` ω `the` n^{th} `root of unity`

Find the bug (10 points)

5. Are these algorithms and/or their proofs correct? Justify your answers (If the algorithm is correct, justify why and if the algorithm is incorrect, either give a counterexample or justify why).

   (a) **Divide and Conquer Algorithm for MST** `MST(`G`: graph on `n` vertices)`

   - $T_1 \leftarrow$ `MST(`G_1`: subgraph of `G` induced on vertices `$\{1,\ldots,n/2\}$`)`
   - $T_2 \leftarrow$ `MST(`G_2`: subgraph of `G` induced on vertices `$\{n/2+1,\ldots,n\}$`)`
   - $e \leftarrow$ `cheapest edge across the cut` $\{1,\ldots,\frac{n}{2}\}$ `and` $\{\frac{n}{2}+1,\ldots,n\}$.
   - `return` $T_1 \cup T_2 \cup \{e\}$.

   Proof of correctness *By the cut property, the cheapest edge e across the cut $\{1,\ldots,\frac{n}{2}\}$ and $\{\frac{n}{2}+1,\ldots,n\}$ belongs to an MST T. On removing the edge e from T, the resulting subtrees must be the minimum spanning trees connecting $\{1,\ldots,\frac{n}{2}\}$ and $\{\frac{n}{2}+1,\ldots,n\}$.*

   (b) **Greedy DFS for shortest paths**
   `Input: Graph` $G=(V,E)$`,a starting vertex` s `and non-negative lengths` ℓ_e `for each edge` $e \in E$.
   `Goal: Compute shortest path to a vertex` v

   `Run DFS, but at each node explore the shortest outgoing edge first until` v `is reached. Return the` s `to` v `path in the DFS tree.`

True or false? Circle the right answer. No explanation needed (15 points)

(No points will be subtracted for wrong answers, so guess all you want!)

1) **T** **F** By starting with number 3 and repeatedly squaring it 1000 times, we can compute $3^{2^{1000}}$ within a day on a laptop.

2) **T** **F** In a graph, if one raises the lengths of all edges to the power 3, the minimum spanning tree will stay the same.

3) **T** **F** $FFT(1, 2, 3, 4) + FFT(-1, -2, -3, -4) = [0, 0, 0, 0]$

4) **T** **F** If $a^{n-1} \equiv 1 \pmod{n}$ for some positive integers $a < n$, then n is a prime.

5) **T** **F** The solution of the recurrence $T(n) = 3T(n/3) + O(n^3)$ is $T(n) = O(n^3)$.

6) **T** **F** Given a polynomial of degree $2^n - 1$, the FFT works by recursively computing 2^n points of two polynomials of degree $2^{n-1} - 1$ and then combining the results.

7) **T** **F** The randomized algorithm to find the median is always faster than running mergesort to find the median.

8) **T** **F** The first edge added by Kruskals's algorithm can be the last edge added by Prim's algorithm.

9) **T** **F** $\log^*(2^n) = 2\log^* n$

10) **T** **F** The heaviest edge in a graph cannot belong to the minimum spanning tree.

11) **T** **F** The longest path in a graph can be computed by negating the cost of all the edges in the graph and then running Bellman-Ford.

12) **T** **F** The maximum spanning tree (spanning tree of maximum cost) can be computed by negating the cost of all the edges in the graph and then computing minimum spanning tree.

13) **T** **F** The family consisting of all possible functions $f : \{1, \ldots, 2^{16}\} \to \{1, \ldots, 256\}$ is a universal hash family.

14) **T** **F** If $\mathcal{H} : \mathbb{Z} \to \{1, \ldots, 170\}$ is a universal hash family of functions, then for every pair of distinct keys x, y there is some function $f \in \mathcal{H}$ such that $f(x) \neq f(y)$.

15) **T** **F** If all edge weights in a graph are either 1 or 2, then the shortest path can be computed in $O(|V| + |E|)$ time.

Semi-Connected Graphs (15 points)

6. A directed graph $G = (V, E)$ is semi-connected if for every pair of vertices u, v either there is a path from u to v or there is a path from v to u or both.

   (a) Give an example of a DAG that is not semi-connected.

   (b) State a necessary and sufficient condition for a DAG to be semi-connected.

(c) Given a DAG, exhibit an algorithm to check if it is semi-connected. (Formal pseudocode is unnecessary, briefly but precisely describe the algorithm and argue its correctness)

Shortest Path with Time-Dependent Edges (20 points)

7. Mr. Albert is on a vacation in Switzerland. There are n cities in Switzerland and m trains $T_1, \ldots, T_m$ between cities. Each train T_i departs city $origin[i]$ at time $dep[i]$ and arrives in city $destination[i]$ at time $arr[i]$. Different trains between the same pair of cities could have different journey times.

   Albert can switch trains at a station instantaneously, i.e., Albert can arrive at time t, switch trains and depart on a train leaving at time t.

   Albert starts his journey at time 0. The arrival and departure times $arr[]$ & $dep[]$ are specified in the units – *"hours from the time Albert started his journey"*.

   (a) Modify Djikstra's algorithm to compute the quickest route to city B starting in city A at time 0. (proof of correctness or running time bound not required)

```
for all cities $v$, $dist[v] = \infty$
$dist[A] = 0$
$H \leftarrow makeQueue()$; // priority queue containing cities with dist values
while $H$ is nonempty
    $v \leftarrow deleteMin(H)$
    for every train $T_i$ from city $v$ do
        (write your pseudocode here)
```

(b) Albert does not mind the train journeys, but really hates waiting at the stations. Albert would like to find an itinerary that reaches B within 3 days, but minimizes the total wait time at the train stations. Design an algorithm to find such a path.

(Hint: Model the problem using a different graph whose nodes specify more than just the city where Albert is. Running Djikstra's algorithm in this graph would give the desired path.)

(Briefly but precisely describe the graph, and argue the correctness of the algorithm.)