CS 170 Midterm 1

Write in the following boxes clearly and then double check.

Name :

SID :

Exam Room :
- ◯ Wheeler 0150
- ◯ Pimentel 1
- ◯ Dwinelle 145
- ◯ Other (Specify):

Name of student to your left :

Name of student to your right :

- **The exam will last 110 minutes.**
- The exam has 12 questions with a total of 120 points. You may be eligible to receive partial credit for your proof even if your algorithm is only partially correct or inefficient.
- Only your writings inside the answer boxes will be graded. **Anything outside the boxes will not be graded.** The last page is provided to you as a blank scratch page.
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Be precise and concise.
- The problems may **not** necessarily follow the order of increasing difficulty.
- The points assigned to each problem are by no means an indication of the problem's difficulty.
- The boxes assigned to each problem are by no means an indication of the problem's difficulty.
- Unless the problem states otherwise, you should assume constant time arithmetic on real numbers. Unless the problem states otherwise, you should assume that graphs are simple.
- If you use any algorithm from lecture and textbook as a blackbox, you can rely on the correctness and time/space complexity of the quoted algorithm. If you modify an algorithm from textbook or lecture, you must explain the modifications precisely and clearly, and if asked for a proof of correctness, give one from scratch or give a modified version of the textbook proof of correctness.
- Assume the subparts of each question are **independent** unless otherwise stated.
- Please write your SID on the top of each page.
- Good luck!

1 Asymptotic Analysis (4 points)

For each pair of functions f and g, specify whether $f = O(g)$, $g = O(f)$, or both.

f	g	$f = O(g)$	$g = O(f)$
$n^3 + \log(n) + 17$	$n^2 + 7\log(n)$	○	○
$n^2 + 4^n$	$n^4 + 2^n$	○	○
$\log(n)$	$\log(n^2)$	○	○
$3^{\log_2(n)}$	n^2	○	○

Solution:

f	g	$f = O(g)$	$g = O(f)$
$n^3 + \log(n) + 17$	$n^2 + 7\log(n)$		x
$n^2 + 4^n$	$n^4 + 2^n$		x
$\log(n)$	$\log(n^2)$	x	x
$3^{\log_2(n)}$	n^2	x	

2 Runtime Analysis (6 points)

Consider the following piece of code:

```
Function what(n) {
      If (n < 2) :
            return;

      what(n/3)
      k = √n
      for i = 1, 2, 3, ..., k {
            for j = 1, 2, 3, ..., k − i {
                  print BLAH
            }
      }
      what(n/3)
      what(n/3)
}
```

Let $T(n)$ denote the runtime of the $what(n)$.

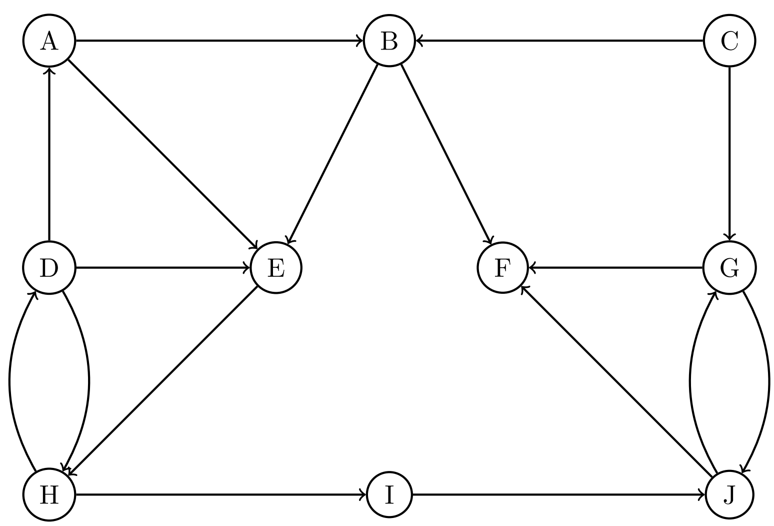
1. Write the recurrence relation for $T(n)$.

   Solution: $T(n) = 3T(n/3) + O(n)$

2. Solve the recurrence relation for $T(n)$ (give the tightest bound $O()$ possible).

Solution: Use Master Theorem with $a = 3$, $b = 3$, $d = 1$ to yield $O(n \log(n))$.

3 Connectivity in Graphs (8 points)



1. Perform DFS in the graph, breaking ties alphabetically, and write down the pre and post numbers.

Vertex v	$pre[v]$	$post[v]$
A		
B		
C		
D		
E		
F		
G		
H		
I		
J		

Exam continues on next page

2. Mark all cross edges if any.

Edge	Fill if cross edge
$A \to B$	◯
$A \to E$	◯
$B \to E$	◯
$B \to F$	◯
$C \to B$	◯
$C \to G$	◯
$D \to A$	◯
$D \to E$	◯

Edge	Fill if cross edge
$D \to H$	◯
$E \to H$	◯
$G \to F$	◯
$G \to J$	◯
$H \to D$	◯
$H \to I$	◯
$I \to J$	◯
$J \to G$	◯
$J \to F$	◯

3. In the following table, list its strongly connected components (SCCs), in the alphabetical order of the smallest vertex contained (e.g. AEF precedes BCD).

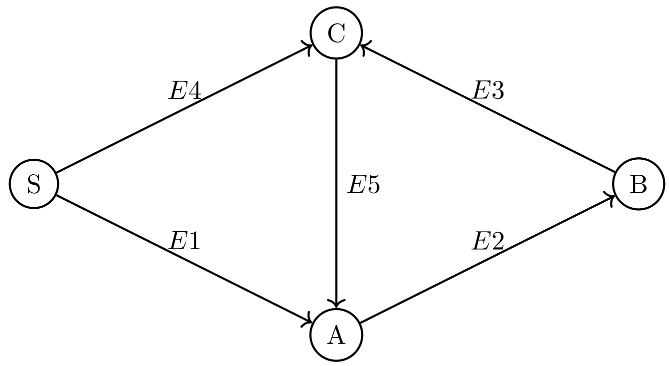
SCC 1		SCC 2	
SCC 3		SCC 4	
SCC 5		SCC 6	
SCC 7		SCC 8	

Solution:

1. A: 1/18; B: 2/17; C: 19/20; D: 5/6; E: 3/16; F: 9/10; G: 11/12; H: 4/15; I: 7/14; J: 8/13
2. Cross edges: FG, CB, CG.
3. ABEHD, C, F, GJ, I

4 Bellman-Ford Algorithm (6 points)

Consider the execution of Bellman-Ford algorithm on the following **directed** graph with **positive** edge weights and the source node S. Edges of the graph are labelled $E1, E2, E3, E4$ and $E5$.



Here is the sequence of update operations carried out by the algorithm.

Iteration Number	Updated Edge
1	E1
2	E2
3	E3
4	E4
5	E5
6	E1
7	E2
8	E3
9	E4
10	E5
11	E1
12	E2
13	E3
14	E4
15	E5

1. What is the earliest iteration after which $dist[A]$ (distance to A) is guaranteed to be correct? If $dist[A]$ is first set to the correct value on iteration x, write x.

2. What is the earliest iteration after which $dist[B]$ (distance to B) is guaranteed to be correct? If $dist[B]$ is first set to the correct value on iteration x, write x.

3. What is the earliest iteration after which $dist[C]$ (distance to C) is guaranteed to be correct? If $dist[C]$ is first set to the correct value on iteration x, write x.

Exam continues on next page

Solution: 5 for A, 7 for B, 4 for C.

5 Minimum Spanning Tree (6 points)

For both subparts of this question, write one edge in each box. Denote an edge with only the vertices, in alphabetical order, and nothing else.
Write: AB, do not write BA, A-B, or AB(9).

1. List the first **six** edges added by Prim's algorithm in the order in which they are added. Assume that Prim's algorithm starts at vertex A and breaks ties lexicographically.

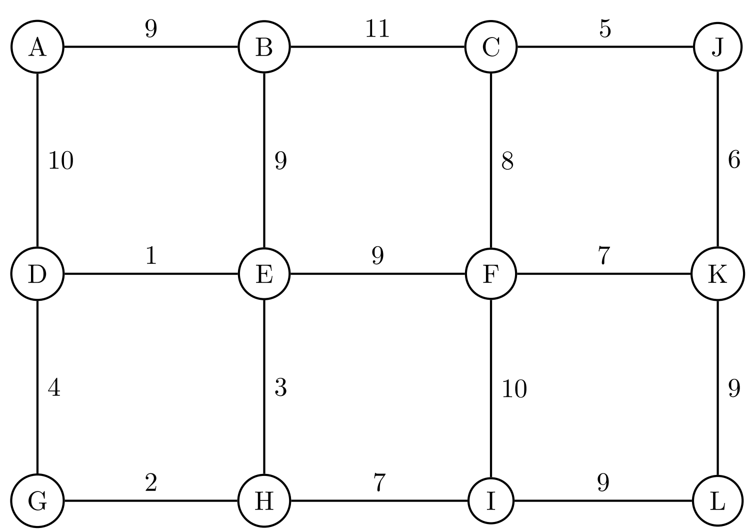
1	2	3	4	5	6

Solution: AB, BE, DE, EH, GH, HI

2. List the first **seven** edges added by Kruskal's algorithm in the order in which they are added. You may break ties in any way.

Solution: (DE, GH, EH, CJ, JK, FK, HI) **or** (DE, GH, EH, CJ, JK, HI, FK)

1	2	3	4	5	6	7



Exam continues on next page

6 Short Answers (18 points)

Note: all subparts are independent from one another.

1. What is the Fourier transform of the vector $[1, 1, 0, 0]$?.

   Solution: $[2, 1 + i, 0, 1 - i]$

2. Let $n > 16$ be a power of 2. Let $\{\omega_1, \omega_2, \omega_3, \ldots, \omega_n\}$ denote all the n^{th} roots of unity. How many distinct numbers are in the set: $\{\omega_1^4, \omega_2^4, \ldots, \omega_n^4\}$?

   Solution: $n/4$.

3. Let $InverseFFT$ denote the inverse Fourier transform. Suppose

$$InverseFFT([a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]) = [0, 0, 2, 0, 0, 0, 0, 0].$$

   What is

$$InverseFFT([a_0^3, a_1^3, a_2^3, a_3^3, a_4^3, a_5^3, a_6^3, a_7^3]) =$$

   Solution: $[0, 0, 0, 0, 0, 0, 8, 0]$.

4. Let $Select(S, k)$ denote the randomized Select algorithm that finds the k^{th} smallest number in a set S.

   Consider the execution of $SELECT(S, n/2)$ on a set S of size n. Let R denote the number of pivots chosen by the algorithm before it terminates.

   (a) In the best case, the value of R (up to constant factors) =

   (b) In the worst case, the value of R (up to constant factors) =

   (c) The expected value of R (up to constant factors) =

   Solution:

   (a) 1
   (b) n
   (c) $\log n$

Exam continues on next page

5. The greedy algorithm on a HornSAT instance returns the following assignment:

$$x_1 = True, x_2 = False, x_3 = True, x_4 = False, x_5 = True$$

For each of the following clauses, indicate whether adding it will necessarily make the instance unsatisfiable. (i.e., there exists no assignment that satisfies all the original clauses *and* the new added clause) Each sub-part below is independent of the other.

(a) $x_1 \implies x_2$ ◯ may be satisfiable ◯ necessarily unsatisfiable

(b) $\overline{x_1} \vee \overline{x_3}$ ◯ may be satisfiable ◯ necessarily unsatisfiable

(c) $\overline{x_5}$ ◯ may be satisfiable ◯ necessarily unsatisfiable

Solution:

(a) may be satisfiable

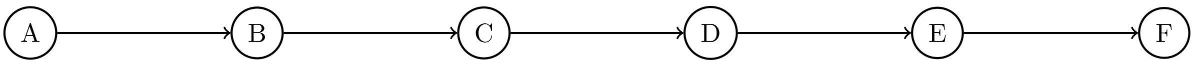
(b) necessarily unsatisfiable

(c) necessarily unsatisfiable

Exam continues on next page ⟹

7 DFS Traversal (5 points)

The DFS traversal of a graph depends on the order in which the vertices are chosen. Consider the following graph:



Suppose we execute a DFS traversal of the above graph, choosing the vertices in arbitrary order (not necessarily lexicographic). Mark each of the following outcomes as possible or impossible.

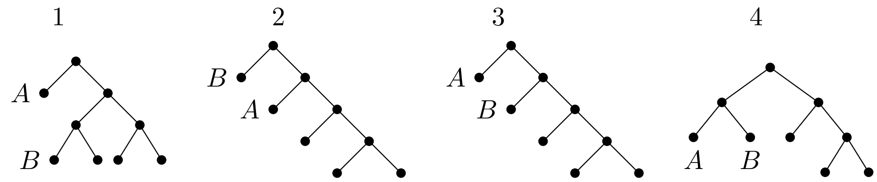
1. $pre[A] < pre[B] < pre[C] < pre[D] < pre[E] < pre[F]$ ◯ possible ◯ impossible
2. $pre[A] > pre[B] > pre[C] > pre[D] > pre[E] > pre[F]$ ◯ possible ◯ impossible
3. $post[A] > post[B] > post[C] > post[D] > post[E] > post[F]$ ◯ possible ◯ impossible
4. $post[A] < post[B] < post[C] < post[D] < post[E] < post[F]$ ◯ possible ◯ impossible
5. $pre[F] < pre[C] < pre[D] < pre[E] < pre[A] < pre[B]$ ◯ possible ◯ impossible

Solution: 1, 2, 3, 5 are possible. 4 is impossible.

Exam continues on next page ⟹

8 Huffman (6 points)

Assume we have a length 100 string consisting of the characters $\{A, B, C, D, E\}$. We know the string contains 30 A's and 40 B's.



For each of the trees shown above, indicate whether the tree is a possible Huffman encoding for some choice of frequencies of other characters C, D and E. Unlabelled leaves may represent any character.

- Tree 1 ◯ possible ◯ impossible If impossible, justify your answer below.

- Tree 2 ◯ possible ◯ impossible If impossible, justify your answer below.

- Tree 3 ◯ possible ◯ impossible If impossible, justify your answer below.

- Tree 4 ◯ possible ◯ impossible If impossible, justify your answer below.

Exam continues on next page

Solution: Only tree 2 is possible. Trees 1 and 3 are impossible because B has a larger frequency than A. The sum of the frequencies of the other characters is 30; therefore, they will all be combined into one subtree and they will then be merged with A. This doesn't happen in tree 4, therefore it is impossible.

Exam continues on next page

9 Legoland (12 points)

Legoland is open for $2n$ days in the summer, and visitors arrive at the park for the first n days. On day i, exactly a_i visitors are arrive at Legoland.

Visitors stay in Legoland for different lengths of time. More precisely, among visitors arriving on any given day, a p_t-fraction of visitors will leave after t-days at Legoland (ie they will spend t days at Legoland then leave the next day).

In this problem, we will devise an algorithm for the following problem:

Formal description:

Input:

1. Number of arrivals $\{a_1, \ldots, a_n\}$.
2. $\{p_1, \ldots, p_n\}$ where p_t is the fraction of visitors that will spend t days.

Output: Determine the number of visitors leaving the park on each day.

---

1. Write down a formula for the number of visitors leaving the park on day ℓ, as a function of $\{a_1, \ldots, a_n\}$ and $\{p_1, \ldots, p_n\}$.

2. Give a succinct and precise description of the algorithm. (Your algorithm should be asymptotically faster than $O(n^{1.5})$. Proof of correctness not required.)

3. What is the runtime of your algorithm?

Solution:

1. The number of visitors leaving the park on day ℓ is $\sum_{1 \leq i < \ell} a_i \cdot p_{\ell - i}$, where we define a_i and p_i to be zero whenever $i > n$.
2. View the input arrays as coefficients of the polynomials $A(x) = \sum_{i=1}^{n} a_i x^i$ and $P(x) = \sum_{i=1}^{n} p_i x^i$. Use FFT to compute coefficients of the degree $2n$ polynomial $A(x) \cdot P(x) = \sum_{i=0}^{2n} c_i x^i$. The number of visitors leaving the park on day ℓ is $c_\ell = \sum_{i+j=\ell} a_i \cdot p_j = \sum_{0 \leq i \leq \ell} a_i \cdot p_{\ell - i}$ by part 1, so we can just return $c_1, c_2, \ldots, c_{2n}$.
3. Runtime is $O(n \log n)$ due to FFT.

10 Faulty Network (15 points)

The road network in the city of Degradia is represented by an undirected graph $G = (V, E)$. Every road degrades over time until it becomes unusable. More precisely, for each edge $(u, v) \in E$ in the network, there is a time t_{uv} after which the road is unusable. Assume all roads start degrading at the same time.

Given this information, devise an efficient algorithm to find the first time at which the network disconnects.

Formal description:

Input:

- An undirected graph $G = (V, E)$
- A positive time $t_{uv} > 0$ for each edge (u, v). The edge (u, v) is removed from the graph G at time t_{uv}.

Output: The first time at which the graph G becomes disconnected.

Observe that if at time t, the network is disconnected, the network will stay disconnected for all times greater than t. Conversely, if at time t, the network is connected, the network will be connected for all times less than t.

Devise an efficient algorithm to determine the first time at which the network disconnects and provide the runtime. (No proof needed. Your algorithm should run in time asymptotically smaller than $O((|V| + |E|)^{1.5})$ to receive full credit.)

Solution:

Method 1: Sort the edges in ascending order of t_{uv}. Say the sorted edges are $e_1, e_2, \ldots, e_m$ where $m = |E|$. The main property is that if the graph on edges $e_i, \ldots, e_m$ is disconnected, then the graph on $e_{i+1}, \ldots, e_m$ is also disconnected. So we can use binary search to find the smallest index i such that the graph on edges $e_i, \ldots, e_m$ is disconnected. At each step, perform DFS/BFS to check whether the graph is connected or disconnected.

Binary search takes $O(\log |E|)$ iterations, and each iteration requires one DFS, which takes $O(|V|+|E|)$ time. So the overall runtime is $O((|V|+|E|)\log |E|)$.

Method 2: Find the maximum spanning tree $T_{\max}$ of the graph where the edges have weight function $w(u,v) = t_{uv}$. We can compute the max spanning tree in multiple ways:

- Modify Kruskal's algorithm so that we initially sort the edges in descending order according to weight.
- First negate all the edges of G to create G_{neg}, and then compute the MST on that graph using Kruskal's. The MST on G_{neg} corresponds to the maximum spanning tree of G.

Then, we output the weight of the lightest edge in $T_{\max}$, which represents the first time at which the graph G becomes disconnected.

Computing the maximum spanning tree takes $O(|E|\log |V|)$, as the modification to Kruskal's adds only $O(|E|)$ extra time (which is dominated by sorting the edges). Then, determining the lightest edge takes $O(1)$ time because you can keep track of it during Kruskal's. Thus, the overall runtime is $O(|E|\log |V|)$.

11 Martian Colonies (14 points)

The Story: (*Feel free to skip the story if you prefer a formal problem description.*)

There are n locations on Mars suitable for building colonies. Let $D[i,j]$ denote the distance between the i^{th} and j^{th} location.

SpaceX needs to select a subset of locations to build colonies at. For safety reasons, each colony must be within a distance R of two other colonies. Devise an algorithm to identify the largest subset of locations to build colonies at.

Formal description:

Input:

- There are n locations numbered $\{1,\ldots,n\}$
- Distances $D[i,j]$ between all pairs of locations i and j in $\{1,\ldots,n\}$.
- A positive number R.

Output: The largest subset $S \subset \{1,\ldots n\}$ of locations such that for each $i \in S$, there exists two other locations $j,k \in S$ such that $D[i,j] < R$ and $D[i,k] < R$.

Give a succinct and precise of an algorithm. (Your algorithm should run asymptotically faster than $O(n^5)$ time. No proof or runtime analysis needed.)

Solution:

Create a graph consisting where each location is a node and for each pair of locations (u, v) the edge (u, v) is added to the graph if $D[i, j] < R$. Start of with $S = V$. While there is a vertex in S adjacent to less than 2 other vertices in S, remove that vertex from S. Output the set S that remains after the while loop terminates.

For correctness, let S^* be the true largest subset satisfying the condition. Notice that our algorithm can never remove a vertex in S^* at any iteration. To see this, consider the first iteration during which we removed a vertex in S^*, say v. Since S^* satisfies the condition, there are two other vertices $v', v'' \in S^*$ that are adjacent to v. Since v was the first vertex in S^* to be removed from S, v' and v'' were still in S at that point, so v couldn't have been removed from S during that iteration.

Runtime is $O(n^3)$ – building the graph takes time $O(n^2)$, and a naive implementation of each iteration takes time $O(n^2)$ per iteration.

Exam continues on next page

12 Mobile Towers (20 points)

Devise algorithms for the following tasks given the input below.

Formal description:

Input:

- An undirected graph $G = (V, E)$ and positive edge weights $\{w_e\}_{e \in E}$ on the edges.
- A subset $T \subset V$ of vertices, referred to as "towers".
- A positive number R

Note: A vertex v is said to be *covered by a tower* w, if the length of the shortest path from v to w is at most R.

---

1. (10 points) List all vertices v that are covered by at least one tower in T. Devise an algorithm for this problem using one execution of Djikstra's algorithm.

   (Give a succinct and precise description of the algorithm. Proof of correctness or runtime analysis not required)

Solution: Modify the graph by adding a new start node s, and adding edges between s and every vertex in T with weight 0. Run Dijkstra's starting from s to find the shortest path lengths from s to every vertex in V, and return all vertices with shortest path lengths at most R from s.

Exam continues on next page ⟹

2. (10 points) **Disclaimer: This subpart is particularly difficult. We recommend working on this problem only after you've finished all previous questions.**

   List all vertices v that are covered by at least **two** towers in T. Observe that one can solve the problem using $|T|$ executions of Djikstra's algorithm, one from each tower. Devise an algorithm for the problem with run-time that is asymptotically faster than $O(|T| \cdot ((|V| + |E|) \cdot \log |V|))$ (i.e. faster than $|T|$ executions of Djikstra's algorithm).

   (Hint: What can you learn by using part(1) on a subset of towers?)

   (a) What is the runtime of your algorithm?

   (b) Give a succinct and precise description of the algorithm. (Proof of correctness not required.)

Solution: Label the towers in T by binary strings of length $l = \log |T|$. For each index $i = 1, \ldots, l$, do the following:

(a) Partition the towers T into two sets T_0 and T_1 depending on the i^{th} bit of their labels. That is, $T_0 = \{t \in T : \text{label}(t)_i = 0\}$, and $T_1 = \{t \in T : \text{label}(t)_i = 1\}$.

(b) Run part 1 on T_0 and T_1, and find the intersection of the vertices returned by the two calls. This consists of all vertices that are reachable by a tower in T_0 and a tower in T_1.

Finally, return all vertices found in at least one of the iterations above.

For correctness, consider a vertex v covered by two different towers t_1 and t_2. Since they are different, their labels must differ in at least one bit (say bit i), and v would have been found in iteration i.

For runtime, we just do $2l = O(\log |T|)$ calls on Dijkstra. The recombining part takes $O(|V| \log |T|)$. So the runtime is $O(\log |T|((|V| + |E|) \cdot \log |V|))$.

This page left intentionally blank
for scratch purposes.
This page **will not be graded**.

DO NOT DETACH THIS PAGE.