CS 170 Midterm 1

Write in the following boxes clearly and then double check.

Name :	
SID :	
Exam Room :	◯ Pimentel 1 ◯ Dwinelle 155 ◯ VLSB 2050 ◯ 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 110 points. You may be eligible to receive partial credit for your proof even if your algorithm is only partially correct or inefficient.
- Only your writing 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 black box, 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 Big-O (6 points)

Label each of the following statements as "True" or "False".

(a) $(\log\log n)^{100} = O(\sqrt[7]{\log(n)})$. ◯ True ◯ False

(b) $2^{\sqrt{\log n}} = O(n^{0.003})$. ◯ True ◯ False

(c) $2^{(\log n)^2} = O(n^{0.003})$. ◯ True ◯ False

(d) $2^{\sqrt{n}} = O(n^{\log n})$. ◯ True ◯ False

(e) If $T(n) = 2T(\frac{n}{2}) + O(\sqrt{n})$, then $T(n) = O(n)$. ◯ True ◯ False

(f) If $T(n) = 2T(\frac{n}{2}) + O(n)$, then $T(n) = \Theta(n\log n)$. ◯ True ◯ False

Solution: For convenience, we write $f(n) = \omega(g(n))$ if, for any constant $c > 0$, $f(n) > c \cdot g(n)$ for all sufficiently large n. Observe that if $f(n) = \omega(g(n))$, then $f(n)$ is not $O(g(n))$.

(a) True. Substituting $t = \log n$ into $(\log t)^{100} = O(t^{1/7})$ yields $(\log\log n)^{100} = O(\sqrt[7]{\log n})$.

(b) True. Substituting $t = \sqrt{\log n}$ into $2^t = O(2^{0.003t^2})$ yields $2^{\sqrt{\log n}} = O(n^{0.003})$.

(c) False. Since $(\log n)^2 = \omega(\log n)$ and $n^{0.003} = 2^{0.003\log n}$, it holds that $2^{(\log n)^2} = \omega(n^{0.003})$.

(d) False. Since $n^{\log n} = 2^{(\log n)^2}$ and $\sqrt{n} = \omega((\log n)^2)$, it holds that $2^{\sqrt{n}} = \omega(n^{\log n})$.

(e) True. Apply the master theorem.

(f) False. Part (e) shows that $T(n)$ can be $O(n)$, in which case $T(n)$ is not $\Theta(n\log n)$.

Exam continues on next page $\Longrightarrow$

2 Recurrences (5 points)

Consider the recurrence relationship $T(n) = 2T(\frac{n}{2}) + O(\frac{n}{\log n})$ with the base case $T(1) = 1$. Prove that $T(n) = O(n \log\log n)$ using the **tree method**.

Hint: You may use the fact that $\sum_{j=1}^{k} \frac{1}{j} = O(\log k)$.

Exam continues on next page $\Longrightarrow$

Solution: For a formal treatment, we let C be the appropriate constant such that

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + C \cdot \left(\frac{n}{\log(n)}\right).$$

There are 2^i subproblems in the ith level of the tree, and each subproblem in the ith level of the tree is of size $n/2^i$. Therefore, the cost at the ith level of the tree is

$$2^i \cdot C \cdot \frac{n/2^i}{\log(n/2^i)} = \frac{Cn}{\log(n) - i}.$$

Since level i of the tree has subproblems of size $n/2^i$, the depth of the tree is $\log_2(n)$. Summing over all levels of the tree, we have

$$\begin{aligned} T(n) &\leq 1 + C \sum_{i=0}^{\log_2(n)-1} \frac{n}{\log(n) - i} && (1) \\ &= 1 + Cn \sum_{j=1}^{\log_2(n)} \frac{1}{j} && (2) \\ &= O(n \log\log n), && (3) \end{aligned}$$

where from (1) to (2) we made the substitution $j = \log(n) - i$, and from (2) to (3) we used the identity in the hint.

Exam continues on next page ⟹

3 Short Answers (10 points)

1. **Fill in True or False:** The following algorithm correctly computes single-source shortest paths in any graph with no negative-length cycles.

   ◯ True ◯ False

Algorithm 1 Input: Weighted Graph $G = (V, E)$; Source Vertex $s \in V$

```
$dist \leftarrow [+\infty, \ldots, +\infty]$
$dist[s] \leftarrow 0$                    ▷ $dist$ is a length $|V|$ array with $+\infty$ everywhere except $s$
for $(u, v) \in E$ do
    for $i = 1, \ldots, |V| - 1$ do
        $dist[v] \leftarrow \min(dist[v], dist[u] + \ell(u, v))$
```

2. **Fill in True or False:** In an undirected graph, it is possible for two vertices with an edge between them to have the same parent in the DFS tree.

   ◯ True ◯ False

3. Recall the greedy algorithm we saw in class for solving the Horn-SAT problem. Suppose that we ran this algorithm on an instance of the Horn-SAT problem with variables a, b, c, d, e and it returned the following satisfying True/False assignment to these variables: $(a, b, c, d, e) = (T, F, F, T, T)$.

   For each of the following True/False assignments, state whether it is possible or impossible for that assignment to also satisfy the same Horn-SAT formula.

   (a) $(a, b, c, d, e) = (T, F, F, T, F)$ ◯ Possible ◯ Impossible

   (b) $(a, b, c, d, e) = (T, T, F, T, T)$ ◯ Possible ◯ Impossible

4. Suppose you have a set of 8 distinct numbers $a_1, \ldots, a_8$ such that the set $a_1^2, \ldots, a_8^2$ has size 4. Are the a_i's necessarily the 8-th roots of unity? If not, provide a counter-example.

   ◯ Yes ◯ No

5. Let $\mathsf{Roots}_4 := \{\omega_0, \omega_1, \omega_2, \omega_3\}$ be the 4-th roots of unity, where $\omega_0 = 1$ and the other three are numbered counter-clockwise on the unit circle. We saw in class that ω_1 is a *generator* of this set, meaning that for every a, there is an integer b such that $\omega_a = \omega_1^b$. Are any of the other 4th-roots of unity generators of Roots_4? If so, list them.

   ◯ Yes ◯ No

Exam continues on next page ⟹

Solution:

1. False. Algorithm 1 is exactly the Bellman-Ford algorithm, except the two for loops have been interchanged. This will result in an incorrect algorithm.

   To see why this results in an incorrect algorithm, note that the for loop over the variable i might change $dist[v]$ when $i = 1$, but it will not change $dist[v]$ any further for $i = 2, \ldots, |V| - 1$. Hence, these iterations do not do anything. As a result, we can replace the for loop over i with the single command
   $$dist[v] \leftarrow min(dist[v], dist[u] + \ell(u, v)).$$
   Writing it this way, we can see that the algorithm just computes the first iteration of Bellman-Ford, which will not correctly compute SSSP on every graph (since Bellman-Ford requires $|V| - 1$ iterations in general).

2. False. As mentioned in lecture, cross edges do not exist in undirected graphs.

3. Note that all variables set to true by the greedy algorithm must be true in all satisfying assignments. However, the variables set to false by the greedy algorithm could potentially be set to true in other satisfying assignments. Thus:

   (a) Impossible

   (b) Possible

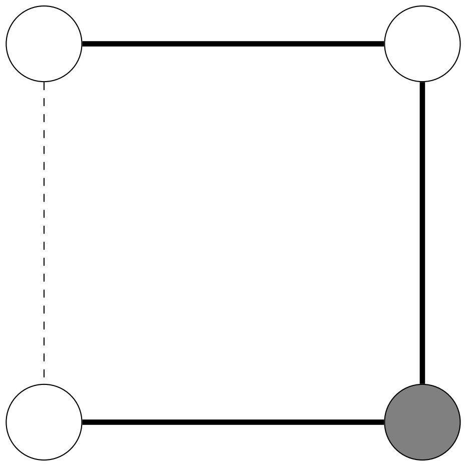
4. No. For example, you could have $-1, 1, -2, 2, -3, 3, -4, 4$.

5. Yes. ω_3 is another generator.

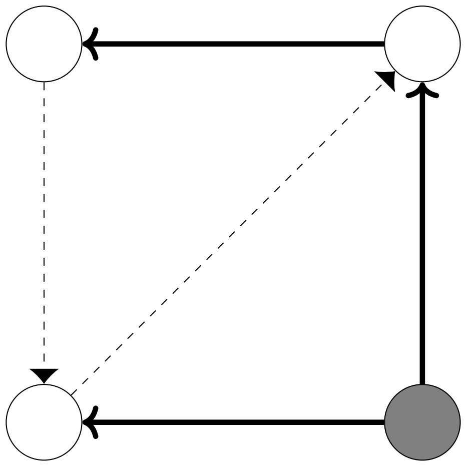
Exam continues on next page ⟹

4 Depth-First Search (10 points)

In the following graphs, the bold solid edges correspond to tree edges in the DFS traversal of the graph and the dashed edges are all of the other edges. The grey, filled-in vertex is the first node explored by the DFS. For each graph, state whether it is possible or impossible for the DFS algorithm to match such a traversal, given some tie-breaking rule between the vertices. Note that the first graph is undirected while the rest are directed.

1. 

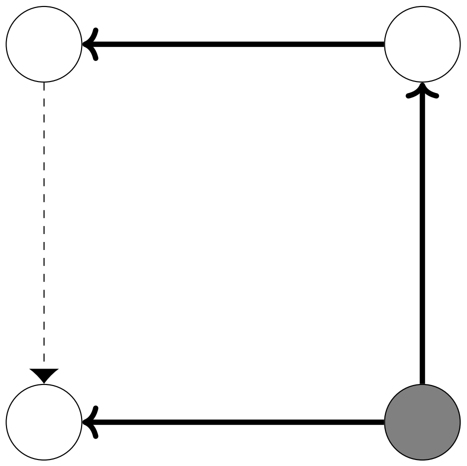
○ Possible ○ Impossible

2. 

○ Possible ○ Impossible

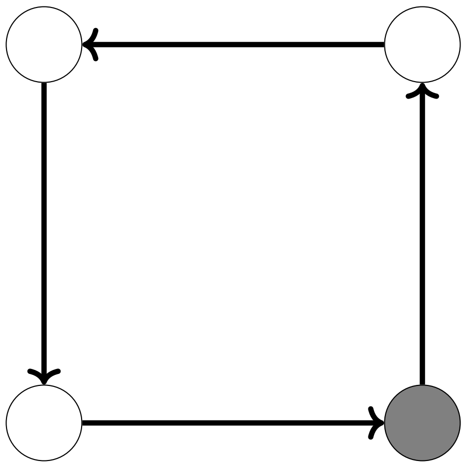
Exam continues on next page

3.



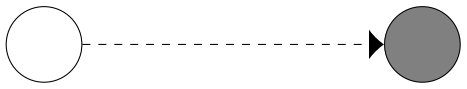
◯ Possible ◯ Impossible

4.



◯ Possible ◯ Impossible

5.



◯ Possible ◯ Impossible

Solution:

1. Impossible. Note that the dotted edge represents a cross edge. Since the graph is undirected, the DFS tree should not have any cross edges in its traversal.
2. Impossible. Note that the dotted edges represent cross edges going in two directions between two branches of the DFS tree. However, in directed graphs we know that a cross edge can only exists from a later branch to an earlier branch (not from an earlier branch to a later branch). Thus, one of the cross edges is impossible to have.
3. Possible. The traversal order will first explore the lower left vertex, before starting a new branch

Exam continues on next page

of exploring the top right and top left nodes.

4. Impossible. This is not a tree.

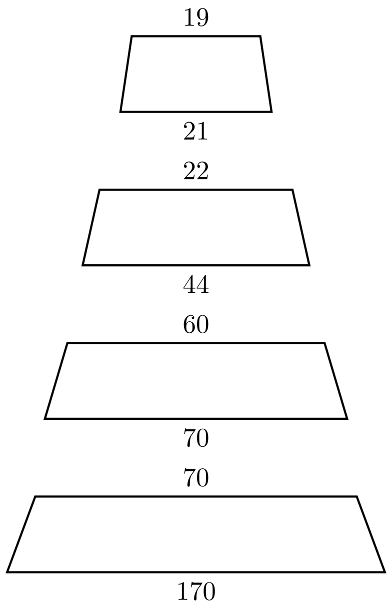
5. Possible. The DFS starts at the right node. Then, a second call to explore is made on the left node.

Exam continues on next page $\Longrightarrow$

5 Spaceship (10 points)

To escape being turned into a paperclip by a rogue artificial intelligence, you are building a spaceship to Mars. You have n trapezoidal components available to build your spaceship. The i-th component is a trapezoid with height 1 and bases of length a_i and b_i such that $a_i < b_i$. A spaceship consists of a sequence of components of decreasing width stacked on top of each other. That is, the i-th component can go on top of the j-th component if and only if $b_i \leq a_j$. Design an efficient algorithm to output the height of the tallest spaceship you can make from the available components.

In the figure below, the spaceship has height 4. (Figure not to scale.)



Give a succinct and precise description of an algorithm to solve this problem. (Proof of correctness not required.) Your algorithm should run in time asymptotically faster than $O(n^3)$ to receive full credit.

Exam continues on next page

Solution: Note that this problem is essentially the same as the interval scheduling problem that we saw in class, and it can be solved by the same greedy algorithm.

Algorithm Description: Begin by sorting the trapezoids in decreasing order of a_i's. This takes $O(n\log(n))$ time. The result is a list of n trapezoidal components

$$(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$$

with $a_1 \geq a_2 \geq \cdots \geq a_n$. Now, assemble the spaceship using the following greedy rule: while the list is non-empty, add the first trapezoid (which has the largest a_i in the list) to the solution. Then remove any trapezoid j from the list for which $b_j > a_i$, and repeat.

This can be implemented in $O(n)$ time as follows: maintain a "width" w which starts at ∞ and records the smallest a_j added to the spaceship so far. Now, scan the list from $i = 1$ to n. On (a_i, b_i): if $b_i \leq w$, add (a_i, b_i) to the spaceship, and set $w = a_i$. Otherwise, do not add it, and move on to component $i + 1$.

Correctness (not required for the exam): This problem is equivalent to interval scheduling. Below we provide a proof of correctness from scratch.

Let $i_1, i_2, \dots, i_k$ be the indices of trapezoids added by the aforementioned algorithm.
Inductive hypothesis: For m, there is an optimal solution that agrees with greedy's solution on the first m indices $i_1, \dots, i_m$.
Base case: $m = 0$ is trivial.
Inductive step: Let $j_1, \dots, j_k$ be the indices of trapezoids in an optimal solution given by the inductive hypothesis for m. Assume without loss of generality that the trapezoids are listed also in decreasing order of their a_js (listed from the bottom of the rocket to the top). We have that $i_1 = j_1, \dots, i_m = j_m$ by the inductive hypothesis. If $i_{m+1} \neq j_{m+1}$, then consider the rocket made of the following parts:

$$i_1, \dots, i_m, i_{m+1}, j_{m+2}, \dots, j_k.$$

All we need to show is that the above rocket is a valid rocket. Note that $i_1, \dots, i_m, i_{m+1}$ forms a valid rocket because the greedy algorithm removes conflicted pieces at every time step. Moreover, $j_{m+2}, \dots, j_k$ also forms a valid rocket because its a subset of an optimal rocket. It remains to show that $a_{i_{m+1}} \geq b_{j_{m+2}}$. We prove this as follows

$$a_{i_{m+1}} \geq a_{j_{m+1}} \geq b_{j_{m+2}},$$

where the first inequality holds by the choice of the greedy algorithm and the second inequality holds because in the optimal solution j_{m+1} is the piece below j_{m+2}.

Runtime Analysis (not required for the exam): We use $O(n\log(n))$ time to sort the trapezoids and then $O(n)$ time to greedily select trapezoids and remove the conflicted pieces. This yields a runtime of

$$O(n\log n) + O(n) = O(n\log n)$$

Exam continues on next page $\Longrightarrow$

6 Karatsuba's (5 points)

Suppose we run Karatsuba's algorithm (using integers in base 10) on the two integers 5432 and 6789 in order to compute the product 5432×6789. Write down the 3 subproblems that Karatsuba's algorithm will call in its top level of recursion.

Subproblem 1: Run Karatsuba's algorithm on ☐ and ☐ .

Subproblem 2: Run Karatsuba's algorithm on ☐ and ☐ .

Subproblem 3: Run Karatsuba's algorithm on ☐ and ☐ .

For each subproblem, write the smaller of the two numbers on the left. Order the three subproblems according to their leftmost number, from smallest to largest.

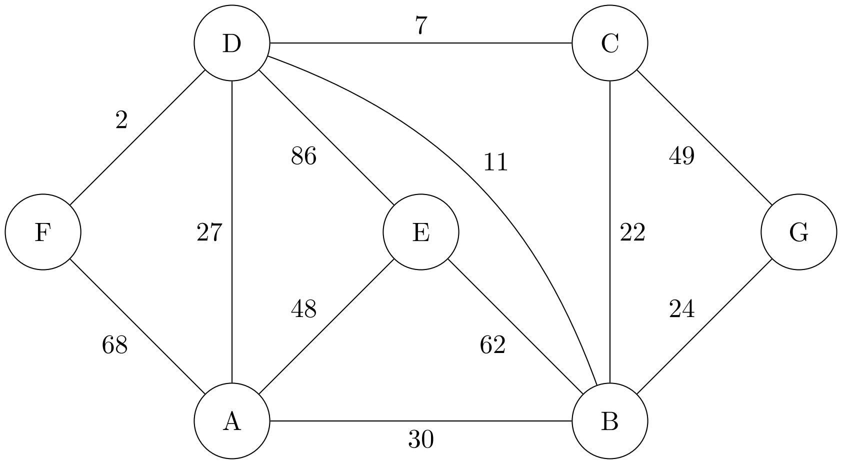
Solution:

Subproblem 1. 32×89
Subproblem 2. 54×67
Subproblem 3. 86×156
For subproblem 3, we want $(54 + 32) \times (67 + 89) = 86 \times 156$.

Exam continues on next page $\Longrightarrow$

7 Kruskal's (6 points)

Run Kruskal's algorithm on the following graph. For each edge, indicate either the order it was added to the MST or if it's not in the MST. For example, if edge UV was the 3rd edge added to the MST, then bubble in "in MST" and write 3 in the corresponding order box. If an edge is not in the MST, bubble in "not in MST" and leave the order box blank.



Edge	In MST?	Order	Edge	In MST?	Order
AB	◯ in MST ◯ not in MST		AD	◯ in MST ◯ not in MST	
AE	◯ in MST ◯ not in MST		AF	◯ in MST ◯ not in MST	
BC	◯ in MST ◯ not in MST		BD	◯ in MST ◯ not in MST	
BE	◯ in MST ◯ not in MST		BG	◯ in MST ◯ not in MST	
CD	◯ in MST ◯ not in MST		CG	◯ in MST ◯ not in MST	
DE	◯ in MST ◯ not in MST		DF	◯ in MST ◯ not in MST	

Solution: The edges that Kruskal's algorithm selects in the order that they are selected:

$$(D, F); (D, C); (D, B); (B, G); (D, A); (A, E).$$

Exam continues on next page $\Longrightarrow$

8 Batched Selection (12 points)

The selection problem asks us to find the k-th largest element of an unsorted list of length n. Recall that QuickSelect solves this problem in expected $O(n)$ time. There is also a deterministic algorithm called DeterministicSelect that solves this problem in $O(n)$ time always. To answer m distinct selection queries for the k_1-th, k_2-th, …, k_m-th largest elements of a given list, one could run DeterministicSelect with $k = k_i$ for each i and take $O(nm)$ time. But we aspire for better!

Design an algorithm to answer m selection queries for distinct $k_1, \ldots, k_m$ on a list $[a_1, a_2, \ldots, a_n]$ of n distinct integers in $O(n \log m)$ time. Your algorithm may call DeterministicSelect as a subroutine. Give a succinct and precise description of your algorithm. (Proof of correctness not required.) Note: algorithms that run in $\Omega(n \log n)$ time will get no credit.

Exam continues on next page

Solution:

a) **Algorithm Description:** Assume without loss of generality that $k_1 < k_2 < \cdots < k_m$. (The k_i can be sorted in $O(m \log m)$ time.) We first find the $k_{\lfloor m/2 \rfloor}$-th largest element a_i with DeterministicSelect in $O(n)$ time. Using a_i as a pivot, we partition $[a_1, a_2, \ldots, a_n]$ into two new lists, L and R, consisting of those elements that are less than a_i and greater than a_i, respectively. List L contains the answers to the queries for the $k_{\lfloor m/2 \rfloor + 1}$-th to k_m-th largest elements, and list R contains the answers to the queries for the k_1-th to $k_{\lfloor m/2 \rfloor - 1}$-th largest elements. We answer the queries for the new lists recursively: on R we run the procedure with queries $k_1, \ldots, k_{\lfloor m/2 \rfloor - 1}$ and on L we run the procedure with *offset queries* $k'_1 = k_{\lfloor m/2 \rfloor + 1} - |R| - 1, \ldots, k'_{m - \lfloor m/2 \rfloor} = k_m - |R| - 1$.

Note that the selection query on value k_i is asking for the k_i-th *largest* element of the list. So if $k_i = 1$, then you want the largest element of the list, not the smallest. This is why L, which contains the smallest elements, also contains the answers to the queries for the largest k_i's. Similarly, this is why R, which contains the largest elements, also contains the answers to the queries for the smallest k_i's.

b) **Runtime Analysis (not required):** Making the recursive call halves the number of selection queries, so our recursion tree will have $O(\log m)$ levels. Drawing out the recursion tree, we see that we spend $O(n)$ time calling DeterministicSelect at each of the $O(\log m)$ levels. This gives us a runtime of $O(n \log m)$, plus the $O(m \log m)$ time it takes to sort the k_i's. Note that since the k_i's are distinct, we have $m \leq n$, and so this is a total runtime of $O(n \log m)$.

Exam continues on next page $\Longrightarrow$

9 Birthday Party Attendance (12 points)

You invited n friends to your upcoming birthday party. Your i-th friend will show up with probability p_i (and does not show up with probability $1 - p_i$) independently of your other friends. Let random variable M be the number of friends who show up. To plan the party, you want to know the probability distribution of M. That is, you would like to compute $\mathbb{P}(M = m)$ for each $m \in \{0, 1, 2, \ldots, n\}$.

a) Describe how to deduce the probability distribution of M from the coefficients of the polynomial

$$P(x) = ((1 - p_1) + p_1 x)((1 - p_2) + p_2 x) \cdots ((1 - p_n) + p_n x).$$

Exam continues on next page

b) Describe an algorithm to compute the coefficients of $P(x)$ in $O(n(\log n)^2)$ time. As a reminder,

$$P(x) = ((1 - p_1) + p_1 x)((1 - p_2) + p_2 x) \cdots ((1 - p_n) + p_n x).$$

Exam continues on next page

Solution:

1. **Deducing M:** For each $m \in \{0, 1, 2, \ldots, n\}$, $\mathbb{P}(M = m)$ is equal to the coefficient of x^m in P.

   Correctness (not required): For each $1 \le i \le n$, define $q_{i,0} = 1 - p_i$ and $q_{i,1} = p_i$. Then

$$\begin{aligned}
P(x) &= ((1-p_1)+p_1x)((1-p_2)+p_2x)\cdots((1-p_n)+p_nx)\\
&= (q_{1,0}x^0 + q_{1,1}x^1)(q_{2,0}x^0 + q_{2,1}x^1)\cdots(q_{n,0}x^0 + q_{n,1}x^1)\\
&= \sum_{z\in\{0,1\}^n} q_{1,z_1}x^{z_1}\cdot q_{2,z_2}x^{z_2}\cdots q_{n,z_n}x^{z_n}\\
&= \sum_{z\in\{0,1\}^n} q_{1,z_1}\cdot q_{2,z_2}\cdots q_{n,z_n}\cdot x^{z_1+z_2+\cdots+z_n}\\
&= \sum_{m=0}^{n} x^m \cdot \Bigg(\sum_{\substack{z\in\{0,1\}^n\\ z_1+\cdots+z_n=m}} q_{1,z_1}\cdot q_{2,z_2}\cdots q_{n,z_n}\Bigg). \qquad (4)
\end{aligned}$$

   Now, interpret $z \in \{0,1\}^n$ as indicating which friends did and did not show up; in particular $z_i = 0$ means that friend i did not showed up and $z_i = 1$ means that they did. Then

$$q_{1,z_1}\cdot q_{2,z_2}\cdots q_{n,z_n}$$

   is the probability that the friends showed up according to the pattern specified by z. Then the coefficient on x^m in Equation (4) simply sums over all attendance patterns z in which m people showed up and computes the probability of z; in other words, it computes the probability that m people show up.

2. We can do divide-and-conquer: recursively compute the two polynomials

$$\prod_{i=1}^{n/2}((1-p_i)+p_ix) \quad \text{and} \quad \prod_{i=n/2+1}^{n}((1-p_i)+p_ix)$$

   and then take their product via FFT in $O(n\log(n))$ time. This satisfies the recurrence

$$T(n) = 2T(n/2) + O(n\log n).$$

   By the tree method, the overall runtime is $O(n(\log n)^2)$.

Exam continues on next page $\Longrightarrow$

10 Unique Topological Sort (10 points)

Given a directed acyclic graph G with n vertices and m edges, design an algorithm that determines whether G has a unique topological sort.

a) Give a succinct and precise description of an algorithm to solve this problem. (Proof of correctness not required.) Full credit is given to algorithms that run in $O(n + m)$ time.

b) What is the runtime of your algorithm?

Exam continues on next page ⟹

Solution:

a) **Algorithm Description:** Compute a topological sort $(v_1, v_2, \ldots, v_n)$ of G. Then, report YES (that G has a unique topological sort) if there is an edge (v_i, v_{i+1}) for all $i < n$ and NO otherwise.

Proof of Correctness (not required): I claim G has a unique topological sort if and only if there is an edge (v_i, v_{i+1}) for all $i < n$. For the "if" direction, note that any other ordering of the vertices would reverse some edge (v_i, v_{i+1}) and thus would not be a topological sort. And for the "only if" direction, consider any topological sort of G. If (v_i, v_{i+1}) is not an edge, then we can swap v_i and v_{i+1} to obtain another topological sort, so the topological sort we found is not unique.

b) The runtime is $O(n + m)$.

Runtime Analysis (not required): A topological sort of G can be computed in $O(n + m)$ time with DFS. Verifying that all edges (v_i, v_{i+1}) exist can be done in $O(n+m)$ time by iterating over each vertex v_i and checking all of its outgoing edges.

Remark. Combining the proof of correctness with Homework 4 Problem 4, we may conclude that G has a unique topological sort if and only if G is semiconnected!

Exam continues on next page $\Longrightarrow$

11 Min-max Tree (12 points)

Let $G = (V, E)$ be an undirected graph with a weight $w_e \geq 0$ for each edge $e \in E$. Recall that a minimum spanning tree T of G is a spanning tree of minimum total weight, where the total weight of T is defined as

$$w(T) = \sum_{e \in T} w_e.$$

In this problem, we will also consider a different type of spanning tree, called a *min-max tree*. A min-max tree T' of G is a spanning tree of G which minimizes the quantity

$$\max(T') = \max_{e \in T'} \{w_e\},$$

among all spanning trees of G. In other words, among all spanning trees of G, the min-max tree T' has as small of a maximum edge as possible.

a) In this part, we will show that if T is a minimum spanning tree, then it is also a min-max tree. We will use a proof by contradiction. To do so, assume for the sake of contradiction that T is *not* a min-max tree of G. Then there is another spanning tree T' of G whose largest weight edge is smaller than T's largest weight edge.

Given T', it is possible to modify T to create a new spanning tree with smaller total weight than T. Give a succinct and precise explanation of how to do so. (Proof of correctness not required. Note that this contradicts the fact that T is a minimum spanning tree.)

Exam continues on next page ⟹

b) In this part, you will show that the converse to the last part is not true: that is, a min-max tree is not necessarily a minimum spanning tree. To prove this, construct a counter-example on a graph G with at most 4 vertices, in which there exists a min-max tree that is not a minimum spanning tree. So you should construct such a graph G, give a min-max tree of G, and show that it is not a minimum spanning tree of G.

Give a concrete construction of G in the following box. Draw the vertices of the graph, the edges between them, and label the weights on the edges.

Give an example of a min-max tree T' of G in the following box. Draw the vertices and only the edges included in the min-max tree. Compute the total weight of the tree $w(T') = \sum_{e \in T'} w_e$.

Give a minimum spanning tree T of G with a total weight $w(T) = \sum_{e \in T} w_e$ which is less than that of the min-max tree you gave in the previous box.

Exam continues on next page $\Longrightarrow$

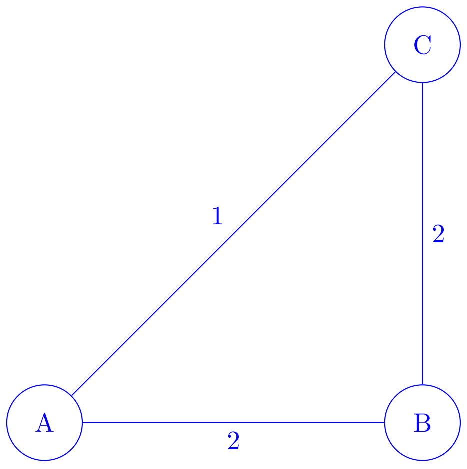
Solution:

(a) Let e be the largest weight edge in tree T. Note that $T \setminus \{e\}$ forms two connected components, U and $V \setminus U$. Since the min-max tree T' is also a spanning tree, some edge in T' connects U and $V \setminus U$. Call this edge e'.

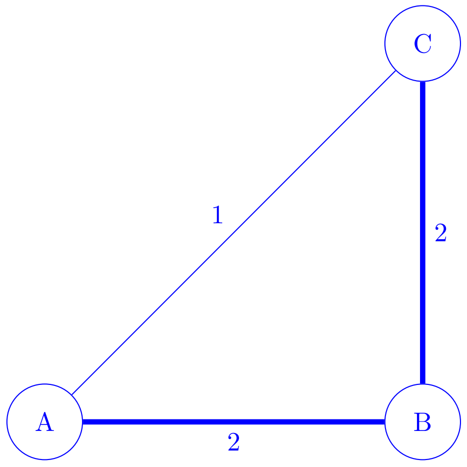
We claim that $T'' = T \setminus \{e\} \cup \{e'\}$ is a spanning tree and $w(T'') < w(T)$.

Proof of correctness (not needed). First note that T'' is indeed a spanning tree because it connects two connected components, U and $V \setminus U$. Moreover, we have that $w_{e'} < w_e$. This is because if T is not a min-max tree, then *every* edge in T' has weight lower than the maximum weight edge of T (i.e., $\max(T') < \max(T)$). Therefore, $w(T'') < w(T)$.

(b) Consider the following graph G:

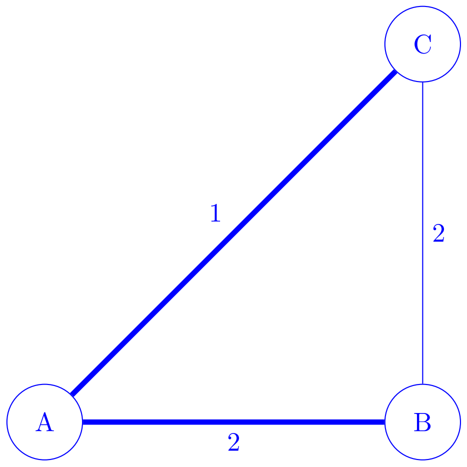


Below, we indicate a min-max tree T' with bolded edges. The weight of this tree is $w(T') = 2 + 2 = 4$.



Notice that this is not an MST! Below, we indicate an MST T with bolded edges. It has weight $w(T) = 1 + 2 = 3 < w(T')$.

Exam continues on next page



Exam continues on next page

12 Flights (12 points)

You are planning to travel by plane from San Francisco to New York. The airline you selected operates in n cities, indexed 1 through n, and has m distinct flight routes. The i-th route flies from city u_i to city v_i and has cost w_i.

Luckily for you, the airline is running a promotion for your trip where you may take up to k flights for free. Your goal is to compute the minimum total cost needed to travel from city 1 (San Francisco) to city n (New York), if you may take up to k flights for free. Give a succinct and precise description of an algorithm to solve this problem. (Proof of correctness not required.) Your algorithm should run in time $O(k(m+n)\log n)$ to receive full credit.

Solution:

Algorithm Description: We create $(k+1)$ copies of the graph. For each $(u_i, v_i) \in E$ and $\ell \in \{1, 2, \ldots, k\}$, add a weight 0 edge from the ℓ-th copy of u_i to the $(\ell+1)$-th copy of v_i. Formally, this

Exam continues on next page ⟹

is the graph $G' = (V', E')$, where

$$V' = \{u_i^\ell \mid \ell \in \{1, \ldots, k+1\} \text{ and } i \in [n]\}$$
$$E' = \{(u_i^\ell, v_i^\ell) \mid i \in [m], \ell \in \{1, \ldots, k+1\}\} \cup \{(u_i^\ell, v_i^{\ell+1}) \mid i \in [m], \ell \in \{1, \ldots, k\}\}.$$

Then, for each $i \in [m]$, let $w(u_i^\ell, v_i^\ell) = w_i$ for all $\ell \in \{1, \ldots, k+1\}$ and $w(u_i^\ell, v_i^{\ell+1}) = 0$ for all $\ell \in \{1, \ldots, k\}$.

Note that G' has $O(kn)$ nodes and $O(km)$ edges. We run Dijkstra's algorithm on G' starting from source node s (i.e. SF) on the first copy, denoted s^1, to compute single-sourced shortest paths in G'. We return $\text{dist}(t^{k+1})$, which represents the distance to the destination node t (i.e. NYC) that uses k free flights.

Runtime Analysis: Running Dijsktra's algorithm on $O(kn)$ nodes and $O(km)$ edges takes $O(k(m+n)\log(kn))$ time. Since $k \leq n$ (or else there is a trivial shortest path with cost 0), this runtime is equal to $O(k(m+n)\log(n))$.

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

DO NOT DETACH THIS PAGE.