CS 6515/4540 (Fall 2025)

Exam 1

Name: _____

GTID: _____

GT Username: _____

INSTRUCTIONS

1. When the exam starts, write your name (or GT ID or GT Username) on each page.
2. Bringing one page of notes is permitted.
3. No books, calculators, or other electronic devices permitted.
4. Make your answers clear and concise.
5. This examination contains **9** pages. Verify no pages are missing.
6. ONLY write on the **FRONT** of each sheet. Backs will not be scanned.
7. You may refer to standard algorithms from class without giving full pseudocode.

I am aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct. I will use no person's help on this test. Also, I have read all the instructions on this page.

Signature:

---

1 Short Answer Questions (35 points)

Unless explicitly mentioned, a short 1–2 line justification is required.

1. (No justification required.) What is the smallest odd number M for which running the Median-of-Medians algorithm with groups of M numbers still yields a linear runtime?

   5

   As discussed in class, 3 does not work as the recursion is not linear time.

2. Given $n \geq 4$, we test primality by checking divisibility for all integers $2 \leq i \leq \lfloor \sqrt{n} \rfloor$. This is a polynomial-time algorithm (in the input size).

   ◯ True

   ◯ False

   False, since n is given in binary representation, this takes exponential time in input size.

3. Suppose $k = \log n$. Then $k^k = \Theta(n)$.

   ◯ True

   ◯ False

   False, $n = O(k^k)$ but not the other way. (Not needed, but the solution to equation $k^k = n$ is $k = \Theta(\frac{\log n}{\log\log n})$.)

4. Let M and M' be two matchings in a bipartite graph G. Consider the symmetric difference $\Delta = M \oplus M'$. Then, Δ may contain a cycle of length 7.

   ◯ True

   ◯ False

   False, all cycles are of even length (as otherwise there is a vertex which has at least two edges incident to it in M or M', which is not possible since they are matchings).

5. Given a weighted graph $G = (V, E)$, let $e = (u, v)$ be the lightest edge incident to u. Then e is in *some* minimum spanning tree of G.

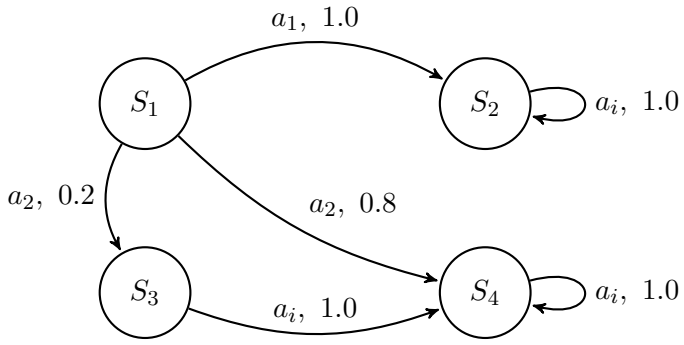
   ◯ True

   ◯ False

   True, by applying the cut property on the cut $(\{u\}, V \setminus \{u\})$

6. Consider the MDP with 4 states $\{S_1, S_2, S_3, S_4\}$ and 2 actions $\{a_1, a_2\}$ as given in the figure. Here, the tuple a, p on an edge means that if we perform action $a \in \{a_1, a_2\}$ at the start state then we end at the destination state with probability p. (When we write a_i in the figure that means this is true for both a_1 and a_2.)

   Moreover, we receive a reward 0 on pulling any action in S_1 or S_4; we receive a reward 10 on pulling any action in S_2; we recieve a reward 100 on pulling any action in S_3. Starting at S_1, what is the optimal policy for horizon $T = 2$? Justify briefly.



   take a2 action.

   The expected reward of a_2 (and then any action) is $0.2 \cdot 100 = 20$, whereas if we took a_1 (and then any action) the expected reward is 10.

7. A PTAS achieves a $(1 - \epsilon)$-approximation in time polynomial in the input size and in $1/\epsilon$.

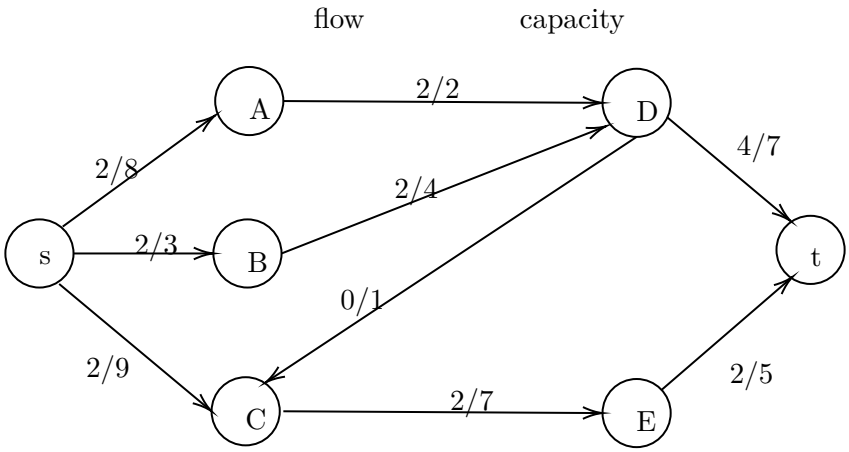
   ◯ True

   ◯ False

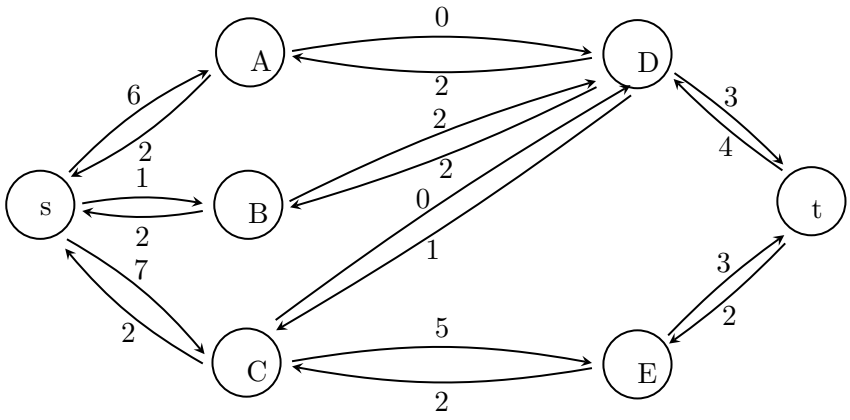
   False, since for PTAS we treat ϵ as a constant, not as part of input. E.g., $n^{1/\epsilon}$ time is allowed for PTAS.

2 Graphs (20 points)

Consider the following st-flow f in a directed graph G, where a/b on an edge denotes flow a and capacity b.

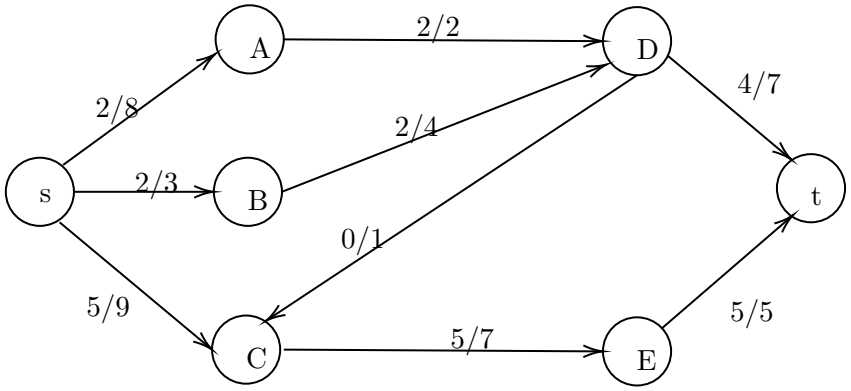


1. (6 pts) Draw the residual graph G^f. Use the convention that for each ordered pair (u, v) you show the residual capacity.



2. (3+3 pts) Find an augmenting path P (or state that none exists) that sends at least 3 units of flow. If P exists, send 3 units of flow along P and draw the resulting flow f'.

   s - C - E - t



(updated flow after sending Δ on P)

3. (4 pts) What is the value of the maximum st-flow in G?

   10

4. (4 pts) List the edges of a minimum st-cut of G.

   (s, B), (E, t), (A, D)

3 Divide and Conquer (25 points)

Given an array $A[0], A[1], \ldots, A[n-1]$, count the number of inversions, i.e., pairs (i, j) with $i < j$ and $A[i] > A[j]$, in $O(n \log n)$ time.

1. (5 pts) Design `MergeAndCount`(X, Y) that takes two **sorted** arrays and returns (i) the merged sorted array and (ii) the number of pairs (i, j) with $X[i] > Y[j]$.
   Example: `MergeAndCount`([1,5],[2,3]) $\rightarrow$ [1,2,3,5] and 2.

Solution:

(a) `MergeAndCount`. Suppose X and Y are sorted arrays. Maintain two pointers i, j starting at 0. While merging into a new array Z:

- If $X[i] \leq Y[j]$, append $X[i]$ to Z and increment i.
- Otherwise ($Y[j] < X[i]$), append $Y[j]$ to Z, increment j, and add $|X| - i$ to the inversion count (since all remaining elements of $X[i..]$ are greater than $Y[j]$).

After one array is exhausted, append the remainder of the other. Return $(Z, \text{cross\_count})$.

2. (10 pts) Using `MergeAndCount` (assume it is implemented), design a divide-and-conquer algorithm to count inversions in A. Give a high-level description of the recursion.

Solution:

(b) Divide and Conquer. To count inversions in A:

1. Split A into two halves $A_{\text{left}}, A_{\text{right}}$.
2. Recursively compute $(A_{\text{left}}^{\text{sorted}}, L)$ and $(A_{\text{right}}^{\text{sorted}}, R)$, where L and R are the inversion counts within each half.
3. Call `MergeAndCount` on the two sorted halves to obtain (A^{sorted}, C), where C is the number of cross inversions.
4. Return $(A^{\text{sorted}}, L + R + C)$.

The recurrence is

$$T(n) = 2T(n/2) + O(n),$$

so the runtime is $O(n \log n)$.

3. (10 pts) Assume $n = 2^k$ and the runtime satisfies

$$T(1) = \Theta(1), \qquad T(n) = 2\,T(n/2) + n \quad (n \geq 2).$$

Prove by induction (do not use Master Theorem) that $T(n) = O(n \log n)$.

Solution: See handwritten notes from Lec 2.

4 Dynamic Programming (20 points)

You are given a knapsack of budget $B \in \mathbb{Z}_{>0}$ and n items, where item i has size $s_i \in \{1, \ldots, B\}$ and value $v_i \in \mathbb{Z}_{>0}$. Choose $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} s_i \leq B$ to maximize $\sum_{i \in I} v_i$.

Design a DP whose running time depends only on n and B (and on $\log v_i$ terms due to bit complexity, which we will ignore for ease of calculations).

1. (5 pts) Define your DP table entries $T(i, b)$, where $i \in \{1, \ldots, n\}$ and $b \in \{1, \ldots, B\}$.

Solution: Let $DP[i, b]$ be the maximum total value achievable using only the first i items with budget at most b (where $0 \leq i \leq n$ and $0 \leq b \leq B$).

2. (8 pts) Give the recurrence in terms of smaller subproblems and briefly justify correctness.

Solution: DP by budget (pseudo-polynomial in B). Define

$$DP[0, b] = 0 \quad (\forall b), \qquad DP[i, 0] = 0 \quad (\forall i),$$

and for $i \geq 1$, $0 \leq b \leq B$ set the recurrence

$$DP[i, b] \;=\; \begin{cases} \max\{DP[i-1, b],\ v_i + DP[i-1,\ b - s_i]\}, & \text{if } s_i \leq b, \\ DP[i-1, b], & \text{if } s_i > b. \end{cases}$$

The answer is $DP[n, B]$. One can recover an optimal set by standard backtracking.

Correctness. We prove by induction on i that $DP[i, b]$ equals the optimal value using items $\{1, \ldots, i\}$ under budget $\leq b$. For $i = 0$ or $b = 0$ the value is 0, which is optimal. For $i \geq 1$: any optimal solution either excludes item i (value $DP[i-1, b]$) or, if $s_i \leq b$, includes it, contributing v_i plus an optimal solution for the remaining budget $b - s_i$ from items $\{1, \ldots, i-1\}$, whose value is $DP[i-1, b-s_i]$ by the induction hypothesis. Taking the maximum yields the optimal value.

3. (7 pts) Analyze runtime in terms of n and B. Is the algorithm polynomial-time (in the input size)?

Solution:

Running time. We fill an $(n+1) \times (B+1)$ table, with $O(1)$ work per cell, so the time is $O(nB)$ and space $O(nB)$ (or $O(B)$ with the common 1D rolling array optimization):

$DP[b] \leftarrow 0 \ \ (\forall b); \qquad \text{for } i = 1 \dots n : \ \text{for } b = B \dots s_i : \ DP[b] = \max\{DP[b],\, v_i + DP[b - s_i]\}.$

Ignoring bit complexity of arithmetic on v_i, the runtime depends only on n and B.

Not a polytime (but pseudo-polytime) algorithm since it depends polynomially on B, instead of the number of inputs bits $\log B$ used to represent B.

END OF EXAM