# CSE 431/531 (Fall 2021) Final Exam (Version B) Solutions

Student Name : _____ Instructor : Shi Li

Student Username : _____ Student ID : _____

| Problem | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Maximum Score | 36 | 6 | 14 | 12 | 12 | 12 | 10 | 10 | 110 |
| Your Score | | | | | | | | | |

The total score of all problems is 112. If you get more than 100 points, your score will be truncated at 100. So, the final score you can get is at most 100.

**Common Notations**

- $\mathbb{R}$: set of real numbers.
- $\mathbb{Z}$: set of integers.
- $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}, \mathbb{Z}_{\geq 0}, \mathbb{Z}_{>0}$ : sets of non-negative real numbers, positive real numbers, non-negative integers and positive integers respectively.
- $[n] = \{1, 2, 3, \cdots, n\}$ for a positive integer $n$.

**Remarks for Problems 4 and 6**

- It is highly recommended that you follow the framework suggested in the problem description, as it will save you a big amount of time. In particular, we are not asking you to optimize the running time: It suffices that your algorithm runs in polynomial time.
- The purpose of examples for the two problems is to help you understand the problems. There may be multiple optimum solutions for the instances. It is possible and OK that your algorithm finds a different optimum solution. So, do not use the given optimum solution to guide your algorithm design.

**Problem 1 (36 Points).** Indicate if each of the following statements is true or false. A true/false answer for each statement is sufficient; you do not need to give proofs/counterexamples for your answers.

(1.1) $\log_2 n = \Theta(\log_{10} n)$. True or False? True

(1.2) Let $f, g : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be two asymptotically positive functions with $f(n) = O(g(n))$. Then $10f(n) + g(n) = O(g(n))$. True or False? True

(1.3) If the recurrence of a running time $T$ is $T(n) = 3T(n/2) + O(n)$, then solving the recurrence using the master theorem gives us $T(n) = O(n^{\log_2 3})$. True or False? True

(1.4) A graph $G = (V, E)$ is bipartite if and only if it does not contain an odd cycle. True or False? True

(1.5) A directed graph $G = (V, E)$ can be topologically sorted if and only if it does not contain a directed triangle. (A directed triangle is a sub-graph with 3 distinct vertices $u, v, w \in V$ and 3 directed edges $(u, v), (v, w)$ and $(w, u)$.) True or False? False

(1.6) Let $G = (V, E)$ be a connected graph with $s \in V$. Let $u$ and $v$ be two vertices in $V$ such that the shortest path from $s$ to $u$ is shorter than that to $v$ (assuming all edges in $E$ have length 1). Then it is guaranteed that $u$ will be visited before $v$ in the Depth-First Search of $G$ starting from $s$. True or False? False

(1.7) Consider an instance for the interval scheduling problem. It is safe to include the shortest job in the final schedule. True or False? False

(1.8) Consider the offline caching problem. The Least-Recently-Used (LRU) algorithm will always give the optimum solution. True or False? False

(1.9) Consider the Huffman-Code instance with 5 letters a, b, c, d, e, and their frequencies are respectively $12, 2, 5, 6, 5$. Then the following mapping gives an optimum prefix encoding of the 5 letters : a $\to$ "0", b $\to$ "100", c $\to$ "101", d $\to$ "110", e $\to$ "111". True or False? True

(1.10) There exists an $O(n)$-time comparison-based sorting algorithm ($n$ is the size of the array to be sorted). True or False? False

(1.11) Quicksort can be implemented as an in-place sorting algorithm. True or False? True

(1.12) Consider the problem of multiplying two single-variable polynomials $p$ and $q$ of degree $n-1$. Then any algorithm that solves the problem must run in time $\Omega(n^2)$ in the worst case, since we have to multiply each coefficient in $p$ with each coefficient in $q$. True or False? False

(1.13) Let $G = (V, E)$ be a connected graph with edge weights $w : E \to \mathbb{R}_{>0}$. Let $e^*$ be the heaviest non-bridge edge in $E$ (assuming $e^*$ exists). Then, there is a minimum spanning tree $T$ of $G$ that does not contain $e^*$. True or False? True

(1.14) Let $G = (V, E)$ be a connected graph with edge weights $w : E \to \mathbb{R}_{>0}$. Let $s$ and $t$ be two vertices in $V$, and let $P$ be a shortest path from $s$ to $t$ in $G$ w.r.t the weights $w$. Then, after we square all the edge weights (that is, change the weight of each $e \in E$ to $(w_e)^2$), $P$ must still be a shortest path from $s$ to $t$ in $G$. True or False? False

(1.15) Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (weights can be positive, 0 or negative). It is guaranteed that $G$ does not contain negative cycles. Then Dijkstra's algorithm can solve the single-source shortest paths problem on $G$. True or False? False

(1.16) Assume P $\neq$ NP and let $X \in$ NP. Then $X$ does not admit a polynomial time algorithm. True or False? True

(1.17) Assume P $\neq$ NP. Then the circuit-satisfiability problem is not in P. True or False? True

(1.18) It is possible that P $\cap$ NP $= \emptyset$. True or False? False

**Problem 2 (4 Points).** Using the definition of the $O$-notation to prove $6(n+3)^3 = O(n^3)$.

*Proof.* For every $n \geq 3$, we have $6(n+3)^3 \leq 6(n+n)^3 = 48n^3$. So, $6(n+3)^3 = O(n^3)$. $\square$

**Problem 3 (14 Points).** We are given a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, using the linked-list representation. You need to design an $O(n + m)$-time algorithm to decide between the following three cases:

- there is no topological-ordering for $G$, in which case your algorithm should output "none",
- there is a unique topological-ordering for $G$, in which case your algorithm should output "unique", and
- there are at least two different topological orderings for $G$, in which case your algorithm should output "multiple".

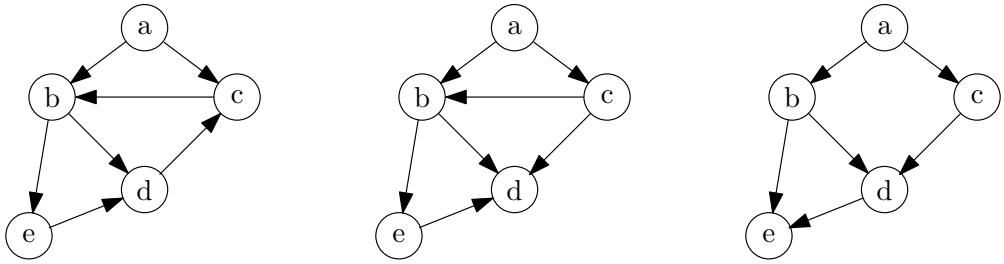



Figure 1: Example input graphs for Problem 3.

For example, consider the three graphs in Figure 1. The outputs for the left-side, middle and right-side graphs are respectively "none", "unique" and "multiple": There is no topological ordering for the left-side graph, there is a unique topological ordering (a, c, b, e, d) for the middle graph, and there are two different topological orderings (a, b, c, d, e) and (a, c, b, d, e) for the right-side graph.

Giving a pseudo-code for your algorithm is sufficient, if the correctness and running time can be easily seen. You can use the topological sort algorithm learned in the class, but you need to describe the main steps of the algorithm, instead of simply saying "using the algorithm learned in the class".

initialize the $d$ array over $V$, so that $d[v]$ is the in-degree of $v$ in $G$
$top \leftarrow 0$, **for** every $v \in V$ **do**: **if** $d[v] = 0$ **then** $top \leftarrow top + 1$, $stack[top] \leftarrow v$
$unique \leftarrow true, sorted \leftarrow 0$
**while** $sorted < n$ **do**
    **if** $top = 0$ **then return** "none"
    **if** $top \geq 2$ **then** $unique \leftarrow false$
    $u \leftarrow stack[top], top \leftarrow top - 1, sorted \leftarrow sorted + 1$
    **for** every outgoing edge $(u, v)$ of $u$ **do**
        $d[v] \leftarrow d[v] - 1$
        **if** $d[v] = 0$ **then** $top \leftarrow top + 1, stack[top] \leftarrow v$
**if** $unique = true$ **then return** "unique" **else return** "multiple"

**Problem 4 (12 Points).** Two teams (call them team A and team B) of chess players need to compete against each other in a tournament. Each team has $n$ players and each of the $2n$ players has a rating. Assume we are in an ideal world where if two players play each other, then the player with the higher rating shall win. $n$ games will be played in the tournament, where in each game a team-A player plays against a team-B player. Each player plays in exactly one game. Your goal is to decide a matching between team A and team B so that team A can win as many games as possible. Given the ratings of the $2n$ players, design an efficient algorithm to solve the problem. You may assume that the ratings of the $2n$ players are all distinct so that there will never be a draw. You need to prove that your algorithm is correct.

For example, assume $n = 5$, the 5 players A1, A2, A3, A4 and A5 in team A have ratings 1500, 2000, 1200, 2200 and 1900 respectively, and the 5 players B1, B2, B3, B4 and B5 in team B have ratings 2300, 2100, 1600, 1100 and 2350 respectively. Then team A can win 3 games in the best

matching: A1 loses to B5, A2 wins B3, A3 wins B4, A4 wins B2, and A5 loses to B1.

Design a greedy algorithm that outputs the maximum number of games team A can win, by following the steps below:

(4a) Assume $n \geq 1$. Design a simple strategy that chooses one player from each team and make them play each other.

(4b) Prove that there is an optimum matching, in which the two players you choose in (4a) play each other.

(4c) After you irrevocably decided to let the two players play each other, what is the residual instance that you need to solve?

(4a) Let $Ai$ be the strongest player in team $A$. If $Ai$ can win some player in team $B$, then let $Ai$ play the strongest player $Bj$ in $B$ whose rating is below $Ai$. Otherwise, let $Ai$ play any player in $B$.

(4b) If $Ai$ can not win any player in team $B$, then team $A$ can win at most 0 games. So letting $Ai$ play any player is safe.

So assume $Ai$ can win some player in team $B$. Then $Bj$ is the strongest player in $B$ who $Ai$ can win. Let $S$ be an optimum matching, that is, a matching in which team $A$ wins the maximum number of games. If $Ai$ plays $Bj$ in the solution $S$, then we are done.

Otherwise assume $Ai$ plays $Bk$ in $S$. Let $Al$ be the player who plays $Bj$ in $S$. We construct a solution $S'$ from $S$ by swapping $Bj$ and $Bk$: in $S'$, $Ai$ plays $Bj$ and $Al$ plays $Bk$.

- If $Ai$ wins $Bk$, then by the choice of $Bj$, $Bj$ is stronger than $Bk$. $Ai$ wins both $Bj$ and $Bk$. So, if $Al$ wins $Bj$, then $Al$ will win $Bk$. So, the number of games team $A$ wins in $S'$ is at least that in $S$.
- Suppose $Ai$ loses to $Bk$. Notice that $Ai$ wins $Bj$. So, in the two games $(Ai : Bk), (Al : Bj)$ in $S$, team $A$ wins at most 1 game. In the two games $(Ai : Bj), (Al : Bk)$ in $S'$, team $A$ wins at least 1 game. So again, the number of games team $A$ wins in $S'$ is at least that in $S$.

So, there is an optimum solution $S'$ in which $Ai$ plays $Bj$.

(4c) Remove the player $Ai$ from team $A$ and the player $Bj$ from team $B$. The residual instance is defined by the remaining players. Rename the players if necessary.

**Problem 5 (12 Points).** Given an array $A$ of $n$ distinct numbers, we say that some index $i \in [n]$ is a local minimum of $A$, if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ (we assume that $A[0] = A[n+1] = \infty$).

Suppose the array $A$ is already stored in memory. Give an $O(\log n)$-time divide-and-conquer algorithm to find a local minimum of $A$. A pseudo-code for your algorithm is sufficient if its correctness and running time can be easily seen.

For example, if $n = 10$ and $A = (60, 30, 20, 10, 80, 50, 40, 70, 100, 90)$, then the indices $4, 7$ and $10$ are all local minimums. It suffices for your algorithm to return one local optimum. So, your algorithm can return $4, 7$ or $10$ for this instance.

1: $l \leftarrow 1, r \leftarrow n$
2: **while** $\ell < r$ **do**
3: $\quad m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$
4: $\quad$ **if** $A[m] < A[m+1]$ **then** $r \leftarrow m$ **else** $\ell \leftarrow m+1$
5: **return** $\ell$

**Problem 6 (12 Points).** Given an array $A$ of $n$ non-negative integers, we need to find the maximum-length increasing subsequence of $A$. That is, we want to find a maximum-length sequence $(i_1, i_2, \cdots, i_t)$ of integers such that $1 \le i_1 < i_2 < i_3 < \cdots < i_t \le n$ and $A[i_1] < A[i_2] < A[i_3] < \cdots < A[i_t]$. For simplicity, you only need to return the length of the longest increasing subsequence of $A$, not the actual subsequence.

For example, if $n = 8$ and $A = (50, 60, 30, 80, 40, 20, 60, 70)$, then the longest increasing subsequence of $A$ is $(A[3] = 30, A[5] = 40, A[7] = 60, A[8] = 70)$, which has length 4.

Design a polynomial time dynamic programming algorithm to solve the problem, by following the steps below:

(6a) Define the cells of the dynamic programming.

(6b) Show how to compute the cells using recursion.

(6c) Specify the order in which you compute the cells.

(6a) For every $i \in [n]$, let $f[i]$ be the length of the longest increasing subsequence of $A$ that ends at $i$.

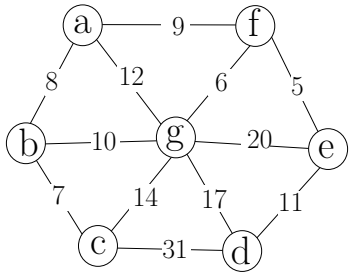
(6b) For every $i \in [n]$, we have

$$f[i] = \max_{j<i:A[j]<A[i]} f[j] + 1,$$

where we assume if there is no $j < i$ with $A[j] < A[i]$, then $\max_{j<i:A[j]<A[i]} f[j] = 0$ (and thus $f[i] = 1$).

(6c) We compute $f[i]$ for every $i$ from 1 to $n$ in the order. The output of the whole algorithm is $\max_{i\in[n]} f[i]$.

**Problem 7 (10 Points).** Use Prim's algorithm to solve the following minimum spanning tree (MST) instance.




You need to use vertex "a" as the root vertex (i.e, the first vertex added to $S$). You can use the following table to describe the execution of the algorithm. For vertices in $v \in S$, $d(v)$ and $\pi(v)$ are irrelevant and thus you can let them be "/". For a vertex $v \notin S$ with $d(v) = \infty$, $\pi(v)$ is also "/". The second and third columns correspond to iteration $i$ indicate the edge added to the MST and the vertex added to $S$ in iteration $i$ respectively.

| $i$ | edge added to MST in iteration $i$ | vertex added to $S$ in iteration $i$ | b | | c | | d | | e | | f | | g | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $d$ | $\pi$ | $d$ | $\pi$ | $d$ | $\pi$ | $d$ | $\pi$ | $d$ | $\pi$ | $d$ | $\pi$ |
| 0 | / | a | 8 | a | $\infty$ | / | $\infty$ | / | $\infty$ | / | 9 | a | 12 | a |
| 1 | (a, b) | b | / | / | 7 | b | $\infty$ | / | $\infty$ | / | 9 | a | 10 | b |
| 2 | (b, c) | c | / | / | / | / | 31 | c | $\infty$ | / | 9 | a | 10 | b |
| 3 | (a, f) | f | / | / | / | / | 31 | c | 5 | f | / | / | 6 | f |
| 4 | (e, f) | e | / | / | / | / | 11 | e | / | / | / | / | 6 | f |
| 5 | (f, g) | g | / | / | / | / | 11 | e | / | / | / | / | / | / |
| 6 | (e, g) | d | / | / | / | / | / | / | / | / | / | / | / | / |

**Problem 8 (10 Points).** For each of the following problems, state (i) whether the problem is known to be in NP, and (ii) whether the problem is known to be in Co-NP. If your answer is yes, you should briefly describe the certifier and the certificate used in the proof.

(8a) Given a directed graph $G = (V, E)$, the problem asks if there is a topological ordering of $G$.
Problem known to be in NP? Yes Problem known to be in Co-NP? Yes

(8b) Given a graph $G = (V, E)$, the problem asks whether $G$ is 3-colorable: Whether we can use 3 colors to color the vertices $V$ in $G$ such that for every edge $(u, v) \in E$, $u$ and $v$ have different colors.
Problem known to be in NP? Yes Problem known to be in Co-NP? No

(8c) A boolean formula is said to be a contradiction if it evaluates to "false" for every assignment of "true/false" values to variables. For example, $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ is a contradiction. (In the formula, "$\vee$", "$\wedge$" and "$\neg$" stand for "or", "and" and "not" respectively.) Given a boolean formula with $n$ variables ($n$ is not fixed), the problem asks whether it is a contradiction.
Problem known to be in NP? No Problem known to be in Co-NP? Yes

(8a) For NP: There is no need to define a certificate. The certifier checks if $G$ can be topologically sorted.

For Co-NP: There is no need to define a certificate. The certifier checks if $G$ can not be topologically sorted.

(8b) For NP: The certificate is a valid 3-coloring of $G$. Given the graph $G$, and a coloring of vertices using 3 colors, the certifier checks if the coloring is a valid 3-coloring.

(8c) For Co-NP: The certificate is an assignment of true/false values to the $n$ variables, for which the formula evaluates to true. Given the formula and the assignment, the certifier checks if the formula evaluates to true for the assignment.