NAME: ____________________________

CSE 421
Introduction to Algorithms
Midterm Exam Spring 2018

Lecturer: Yin Tat Lee

30 April 2018

DIRECTIONS:

- Answer the problems on the exam paper.
- You may use facts proven in class.
- If you need extra space use the back of a page.
- You have 50 minutes to complete the exam.
- Please do not turn the exam over until you are instructed to do so.
- Good Luck!

1	/40
2	/20
3	/20
3	/20
Total	/100

Problem 1 (40 points, 4 each).

For each of the following problems circle **True** or **False**. **NO** justification is needed.

1. $n^{1.2} = O(n2^{\sqrt{\log n}})$. True / False

   Solution. *False.*

2. $n^{10} = O(2^{n^{0.1}})$. True / False

   Solution. *True.*

3. Any graph G with no cycles has exactly $n - 1$ edges. True / False

   Solution. *False. (Only holds for* **connected** *graph with no cycles.)*

4. Dijkstra's algorithm works for directed graph with positive length. True / False

   Solution. *True.*

5. In the Union Find data structure, every tree representing a connected component can have depth at most $O(1)$. True / False

   Solution. *False.*

6. For the stable matching problem, there can be more than 1 solution. True / False

   Solution. *True.*

7. There is a polynomial time algorithm for testing if an undirected graph is 2-colorable or not. . True / False

   Solution. *True.*

8. Let T be a breadth-first search tree of a undirected graph. Let (x, y) be an edge of G that is not an edge of T, then one of x or y is an ancestor of the other. True / False

   Solution. *False. (This is for DFS.)*

9. If $T(n) \leq 2T(n/2) + n$, then $T(n) = O(n \log^2 n)$. True / False

   Solution. *True.* $(n \log n = O(n \log^2 n))$.

10. Let C be any cycle in a graph G with distinct costs, and let edge e be the cheapest edge belonging to C. Then e belongs to all minimum spanning tree of G. True / False

    Solution. *False.*

Problem 2 (20 points). Given a directed graph $G = (V, E)$ with no cycle. Give a $O(|E| + |V|)$ time algorithm to check if there is a directed path that touches every vertex exactly once. Prove the correctness and the runtime of the algorithm.
Hints: You can use the fact that topological sort can be done in $O(|E| + |V|)$ time.

Solution.

Algorithm

- *Find the topological order* $v_1, v_2, v_3, \cdots, v_n$.
- *If there is an edge from* $v_i \to v_{i+1}$ *for all* $i = 1, 2, \cdots, n-1$
  - *Output "YES".*
- *Else*
  - *Output "NO".*

Runtime

The runtime is dominated by the topological sort, which is $O(|E| + |V|)$ *time.*

Correctness

There are two cases: 1) output "YES". In this case, the algorithm indeed finds a path $(v_1 \to v_2 \to \cdots \to v_n)$ *that touches every vertices.*

2) output "NO". In this case, there is i *such that no edge goes from* v_i *to* v_{i+1}*. Therefore, any path touches* v_i *must go to* v_j *for some* $j > i + 1$ *(It cannot go to* v_j *with* $j < i$ *because of the topological order.) However, after the path goes to* v_j *for* $j > i + 1$*, it cannot go back to* v_{i+1} *due to topological order again. Hence, if the path touches* v_i*, it will miss* v_{i+1}*. Therefore, there is no path that touches every vertices.*

In both cases, the algorithm is correct. (Note that the proof handled the disconnected case automatically.)

Problem 3 (20 points). Consider the interval scheduling problem in the class. We have n jobs. The j-th job starts at $s(j)$ and finishes at $f(j)$. Jobs are compatible if they do not overlap. Our goal is to find a maximum subset of compatible jobs.

Instead of selecting the first compatible job to finish as in the class, we consider the greedy algorithm that selects the last compatible job to start. Prove that this yields an optimal solution or give an example to disprove this algorithm.

Solution. *The algorithm yields an optimal solution.*

Correctness

We prove this by induction. Let $i_1, i_2, \cdots, i_k$ be jobs picked by the new greedy method and $j_1, j_2, \cdots, j_m$ be any solution. (We sort the job in reverse chronological order. Namely, $f(j_1) \geq s(j_1) \geq f(j_2) \geq s(j_2) \geq \cdots$.)

Induction statement: "$s(i_r) \geq s(j_r)$"

Base Case: $s(i_1) \geq s(j_1)$ since we select the last job to start.

Inductive step: Since $s(i_r) \geq s(j_r) \geq f(j_{r+1})$, j_{r+1} is among the candidates considered by greedy when it picked i_{r+1}. Since it picks the last job to start, we have $s(i_{r+1}) \geq s(j_{r+1})$.

In particular, induction shows that $s(i_k) \geq s(j_k)$.

If $m > k$, then j_{k+1} is among the set of candidates for i_{k+1}. It is impossible because the algorithm ends only if there is no candidates left. Hence, we have $m \leq k$. Thus, greedy method picks at much as any other solutions.

Problem 4 (20 points).
Given an array of positive numbers $a = [a_1, a_2, \cdots, a_n]$. Give an $O(n \log n)$ time algorithm that find i and j (with $i \le j$) that maximize the subarray product $\prod_{k=i}^{j} a_k$. Prove the correctness and the runtime of the algorithm.

For example, in the array $a = [3, 0.2, 5, 7, 0.4, 4, 0.01]$, the sub-array from $i = 3$ to $j = 6$ has the product $5 \times 7 \times 0.4 \times 4 = 56$ and no other sub-array contains elements that product to a value greater than 56. So, the answer for this input is $i = 3, j = 6$.

Hints: Divide and Conquer.

Solution.

Algorithm

function $(i, j) = \texttt{MAXSUB}(a_1, a_2, \cdots, a_n)$

- *If* $n = 1$
  - *Output* $i = j = 1$.
- *Else*
  - $(i_1, j_1) = \texttt{MAXSUB}(a_1, \cdots, a_{\lfloor n/2 \rfloor})$.
  - $(i_2, j_2) = \texttt{MAXSUB}(a_{\lfloor n/2 \rfloor + 1}, \cdots, a_n)$.
  - *Find* $i_3 \le \lfloor n/2 \rfloor$ *that maximize* $\prod_{k=i_3}^{\lfloor n/2 \rfloor} a_k$.
  - *Find* $j_3 > \lfloor n/2 \rfloor$ *that maximize* $\prod_{k=\lfloor n/2 \rfloor + 1}^{j_3} a_k$.
  - *Compare the subarray product for* (i_1, j_1), (i_2, j_2) *and* (i_3, j_3) *and output the one with the largest subarray product.*

Runtime

The runtime satisfies $T(n) = 2T(n/2) + O(n)$. *So, we have* $T(n) = O(n \log n)$.

Correctness

Induction statement: "The algorithm is correct for input size $\le n$*"*

Base case $n = 1$*: The algorithm is correct because* $i = j = 1$ *is the only possible output.*

Inductive step:

Case 1: $j \le \lfloor n/2 \rfloor$.

The algorithm finds the solution from the first sub-problem (due to the induction hypothesis).

Case 2: $i > \lfloor n/2 \rfloor$.

The algorithm finds the solution from the second sub-problem (due to the induction hypothesis).

Case 3: $i \le \lfloor n/2 \rfloor$ *and* $j > \lfloor n/2 \rfloor$.

Note that $\prod_{k=i}^{j} a_k = \prod_{k=i}^{\lfloor n/2 \rfloor} a_k \times \prod_{k=\lfloor n/2 \rfloor}^{j} a_k$. *Since* i *and* j *maximize the left hand side,* i *must be the maximizer of* $\prod_{k=i}^{\lfloor n/2 \rfloor} a_k$ *and* j *must be the maximizer of* $\prod_{k=\lfloor n/2 \rfloor}^{j} a_k$.

Therefore, the algorithm correctly finds it in this case.