Final Exam CS601, Algorithms and Complexity, Fall 2024

Instructions:

1. **DO NOT OPEN THE EXAM UNTIL YOU ARE INSTRUCTED TO.**
2. There are **five** problems printed across **three** pages in total. Please make sure all of you have exams with all the problems and all the pages.
3. Answer all of the questions as well as you can. You have three hours to complete this exam.
4. The exam is non-collaborative; you must complete it on your own. If you have any clarification questions, please ask the course staff. We cannot provide any hints or help. No cell phone or electronic devices or any use of Internet is permitted in this exam. Indeed, **if you are caught engaging in unseemly practices, severe action will be taken.**
5. This exam is closed-book, except for
   (a) Up to *two double-sided sheets of paper* that you have prepared ahead of time. You can have anything you want written on these sheets of paper.

Good Luck **Total points: 100**
Time Limit: 2 hours

---

1. **(20 pt.) (A tryst with Big-Oh and friends)**

   (a) **(5 pt.)** Prove or disprove: $3^n = O(2^n)$.

   (b) **(5 pt.)** Prove or disprove: $2^{(\log n)^2} = O(n^{1.5})$.

   (c) **(10 pt.)** Let $f(n) = 2^{n!}$ and $g(n) = (2^n)!$. Is $f(n) = O(g(n))$? Briefly explain your answer.

SOLUTION:

(a) Note that $3^n = 1.5^n \cdot 2^n \geq \omega(2^n)$. Disproved.

(b) $2^{(\log n)^2} = n^{\log n} \geq \omega(n^{1.5})$. Disproved.

(c) We take logs of both of these functions f and g. We note $\log f = 2^{n!}$. Also, $\log g = 2^n \log 2^n = n \log 2^n$. One notes that $\log g = o(\log f)$. This means $g = O(f)$.

2. **(20 pt.) (Fun with Induction)**

(a) **(10 pt.)** Recall, the Fibonacci numbers, defined recursively as $F_1 = 1, F_2 = 1$, and $F_n = F_{n-1} + F_{n-2}$. Prove that every third Fibonacci number is even. For example, $F_3 = 2$ is even and $F_6 = 8$ is even.

(b) **(10 pt.)** For any natural number $n \in \mathbb{N}$ and $x > 0$, show that $(1+x)^n \geq 1 + nx$.

SOLUTION:

(a) We will show that for all $n \geq 1$, F_{3n} is even by induction. For the base case, let us consider the case $n = 1$. In this case, we note that $F_{3\cdot 1} = F_3 = 2$ which is even.

For the inductive step, let us inductively assume for all $m \leq n$, we have F_{3m} is even. Now, let us consider the case where $m = n + 1$. We note that

$$\begin{aligned} F_{3n+3} &= F_{3n+2} + F_{3n+1} \\ &= F_{3n+1} + F_{3n} + F_{3n+1} \\ &= 2 \cdot F_{3n+1} + F_{3n} && \text{By induction hypothesis } F_{3n} \text{ is even.} \end{aligned}$$

In all, this means F_{3n+3} is even.

(b) **Base case:** When $n = 0$, we note $(1+x)^0 = 1 \geq 1 + 0 \cdot x$ and thus the base case holds.

Inductive Hypothesis: Assume $(1+x)^k \geq 1 + kx$ for all $n \leq k$ where $k \in \mathbb{N}$.
Inductive Step: For $n = k + 1$, we note

$$\begin{aligned} (1+x)^{k+1} &= (1+x)^k \cdot (1+x) \\ &\geq (1+kx) \cdot (1+x) && \text{By induction hypothesis} \\ &= 1 + kx + x + kx^2 \\ &= 1 + (k+1)x + kx^2 \\ &\geq 1 + (k+1)x \end{aligned}$$

3. **(20 pt.)**

Design a linear-time algorithm for the following task:

Input: A directed acyclic graph G.

Question: Does G contain a directed path that touches every vertex exactly once?

[**HINT:** *Use topological sort*]

SOLUTION: The main idea is to use topological sort as a handy subroutine. This algorithm arranges the vertices in G in a linear order from left to right. To find a path that touches all the vertices exactly once, we just need to check that for every $i \in \{1, 2, 3, \ldots, n-1\}$, there is an edge in this sorted DAG between v_i and v_{i+1}. The time complexity of both of these steps is $O(m+n)$.

4. **(20 pt.) (Median of two sorted arrays)**

Finding the median of a sorted array is easy: return the middle element. But what if you are given two sorted arrays A and B, of size m and n respectively, and you want to find the median of all the numbers in A and B? You may assume that A and B are disjoint.

(a) **(5 pt.)** Give a naive algorithm running in $\Theta(m+n)$ time. [**HINT:** *Feel free to use linear time algorithm for finding medians as a blackbox*]

(b) **(10 pt.)** If $m = n$, give an algorithm that runs in $\Theta(\lg n)$ time.

(c) **(5 pt.)** Give an algorithm that runs in $O(\lg(\min m, n))$ time, for any m and n. [**HINT:** *Say $m = |A| > |B| = n$. A convenient approach is to trim out the elements of A till only n elements remain and now you can find the median of the remaining subarray of A and B.*]

SOLUTION:

(a) Merge the two sorted arrays (which takes $O(m+n)$ time) and find the median using linear-time selection.

(b) Pick the median m_1 for A and median m_2 for B. If $m_1 = m_2$, return m_1. If $m_1 > m_2$, remove the second half of A and the first half of B. Then we get two subarrays with size $n/2$. Repeat until both arrays are smaller than a constant. The case where $m_1 < m_2$ is analogous.

(c) Without loss of generality, assume $|A| = m > n = |B|$. We can safely remove elements $A[0 : \frac{m-n}{2}]$ and $A[\frac{m+n}{2} : m-1]$ because none of these elements can be the median of $A + B$. After this process, we get two arrays of size approximately n. Then we can run part (b). The complexity is $\Theta(\lg(min(m,n)))$.

5. **(20 pt.) (Shortest Paths with colored edges)**

I am given a graph $G = (V, E)$ whose edges come in 3 colors – red, blue, and yellow (which are denoted E_R, E_B, E_Y respectively). I know $E = E_R \cup E_B \cup E_G$. The edge e has cost $c_e \geq 0$.

Devise an efficient algorithm to compute the length of the shortest path from a vertex $s \in V$ to a vertex t that never uses the edges of the same color consecutively along the path. For example, if the path takes a red edge from vertex i to vertex j, then it can't take a red edge out of vertex j.

Also, write out the proof of correctness and also give the running time of your algorithm.

[**HINT:** *It is helpful to start with three copies of the vertex set V. One in red, the other in blue, the last in yellow. You want to now add edges in this empty graph to capture what kind of paths you want. As in, you want to prohibit two consecutive red edges say (i, j) and (j, k). What kind of edges should you add in this auxiliary graph so that running Dijkstra's on this graph does the job?*]

SOLUTION:

(a) As noted in the hint, the key is to make three copies of the vertex set in each of – G_R, G_B, G_Y. I denote the copy of vertex u in G_i as u_i.

For each edge (u, v) in E_R, make a copy between the $u_B tov_R$ and between $u_Y andv_R$. For each edge $(u, v) inE_Y$, make a copy between the u_B to v_T and between u_A and v_T . For each edge (u, v) in E_B, make a copy between the u_R to v_B and between u_Y and v_B. Create a dummy node s_D with edges of weight 0 to $s_R, s_B,$ *and* s_Y . Also create a dummy node t_D with edges of weight 0 to $t_R, t_B,$ *and* t_Y . Run Dijkstra's Algorithm on this newly constructed graph starting at s_D, and return $dist(t_D)$. Intuitively, (WLOG) arriving at a node $v_B in G_B$ means that you took a blue edge to arrive at v, so you need to leave using either a red or a yellow edge; this construction of the graph captures that information.

(b) **Runtime:** $\Theta((m + n) \cdot \log n)$.

(c) **Proof of correctness:** Our auxiliary graph does not have an edge between two vertices in the same color class. Denoting this auxiliary graph as G', this means if I reach some vertex v_B by using a blue edge, then I cannot use a blue edge leaving this vertex!

When constructing our three sub-graphs, every edge we add will have the same weight as its respective edge in G. Therefore any path in our new graph will have the same cost as the path in G visiting the same vertices in the same order and thus respects the colors of the edges along the path in G. Thus this problem can be solved by simply finding the shortest path from any one of $s_R, s_B,$ *or* s_Y to any one of $t_R, t_B,$ *or* t_Y . We add the dummy nodes so we only have to run Dijkstra's once.