Instructions that will appear on the real exam

- **DO NOT OPEN THE EXAM UNTIL YOU ARE INSTRUCTED TO.** (But of course you can do what you want with this practice exam :)
- Answer all of the questions as well as you can. You have **one hour.**
- 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.
- This exam is **closed-book**, except for **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.
- **Please DO NOT separate pages of your exam**. The course staff is not responsible for finding lost pages, and you may not get credit for a problem if it goes missing.
- There are a few pages of extra paper at the back of the exam in case you run out of room on any problem. If you use them, please clearly indicate on the relevant problem page that you have used them, and please clearly label any work on the extra pages.
- Please make sure to sign out of the roster when handing in your completed exam to the teaching team.
- **Please do not discuss the exam until after solutions are posted!** (Practice exam note: Of course feel free to discuss the practice exam solutions with us any time!)

General Advice

- If you get stuck on a question or a part, move on and come back to it later. The questions on this exam have a wide range of difficulty, and you can do well on the exam even if you don't get a few questions.
- Pay attention to the point values. Don't spend too much time on questions that are not worth a lot of points.
- There are **100** total points on this exam. There are **three problems** across **ten pages**.

Name and SUNet ID (please print clearly): SOLUTION

This page intentionally blank. Please do not write anything you want graded here.

Honor Code (this will appear on the real exam)

The Honor Code is an undertaking of the Stanford academic community, individually and collectively. Its purpose is to uphold a culture of academic honesty. Students will support this culture of academic honesty by neither giving nor accepting unpermitted academic aid on this examination.

This course is participating in the proctoring pilot overseen by the Academic Integrity Working Group (AIWG), therefore proctors will be present in the exam room. The purpose of this pilot is to determine the efficacy of proctoring and develop effective practices for proctoring in-person exams at Stanford.

Unpermitted Aid on this exam includes but is not limited to the following: collaboration with anyone else; reference materials other than your cheat-sheet (see below); and internet access. (Of course you can do whatever you want on this practice exam; we suggest that you take it in as close to a test environment as you can though.)

Permitted aid on this exam includes a "cheat-sheet:" two double-sided sheets of paper with anything written on them, which you have prepared yourself ahead of time.

I acknowledge and attest that I will abide by the Honor Code:

[signed] _____

Exam Break Sign-out

I pledge that during my exam break:

- I will not bring any paper, electronic devices (phone, smart watch, smart glasses, etc), or aid (permitted or unpermitted) *out of or into* the exam room.
- I will not communicate with anyone other than the course instructional staff about the content of the exam.

Signature Confirming Honor Code Pledge	Exit Time	Return Time	Proctor Initial	Length (min)

If you are feeling unwell and are not able to complete the exam, please connect with the proctor to discuss options.

Good Luck!

This page intentionally blank. Please do not write anything you want graded here.

1. **(48 pt.)** [**Multiple Choice!**] For each of the parts below, clearly fill in **all** of the answers that are true.

Grading note: In each part below, each of the answers other than "None of the above" are worth points separately. The "None of the above" option on its own is not worth any points, it is there so you can indicate that you intentionally didn't fill in anything. If you fill in both "None of the above" and any other answer, we will ignore your "None of the above". If you don't fill in anything and also don't fill in "None of the above", we will assume you did not complete the problem and you will get a zero. Ambiguously filled-in answers will be marked as incorrect.

[**We are expecting:** *For each part, clearly fill in answers like this:* ■*, or leave them blank like this:* ☐*. No justification is required or will be considered when grading.*]

(a) **(12 pt.)** Which of the following are true?

☐ (A) A Red-Black tree storing n items supports INSERT/SEARCH/DELETE in worst-case time $O(\log n)$.

☐ (B) A Red-Black tree storing n items is always *perfectly* balanced (meaning that the height is at most $\lceil \log n \rceil$).

☐ (C) Suppose you build a hash table using a universal hash family, to store n items with n buckets. If you insert an item x multiple times, it may end up in different buckets each time due to the randomness in the hash family.

☐ (D) Let $\mathcal{H}$ be a universal hash family, consisting of functions that map a universe $\mathcal{U}$ of size $M > n^2$ down to n buckets. Then for all $h \in \mathcal{H}$, there is some $x \in \mathcal{U}$ that collides with $\Omega(n)$ things under h (that is, there are $\Omega(n)$ values of $y \in \mathcal{U}$ with $h(x) = h(y)$).

☐ (E) None of the above

SOLUTION:
(A) is true.
(B) is false. A Red-Black tree has height at most $O(\log n)$ (where the constant in the big-Oh is 2), but not necessarily $\leq \lceil \log n \rceil$.
(C) is false: when we build a hash table from a hash family, we randomly choose one function h, and use it for the whole time. We don't re-choose h between different INSERTs.
(D) is true: this follows from the pigeonhole principle, and/or our discussion in class about worst-case hash tables.

(b) **(12 pt.)** Which of the following always take time $O(n)$?

☐ (A) Using RadixSort to sort n integers between 1 and 2^n with base n.

☐ (B) Running DFS on an arbitrary graph.

☐ (C) Running BFS on a graph with degree at most 7.

☐ (D) Running in-order traversal on a Binary Search Tree to output the elements in sorted order.

☐ (E) None of the above.

SOLUTION:
(A) is false, since the maximum size of the integers is too big. (The running time is only guaranteed to be $O(n \log_n(2^n))$, which is $O\left(\frac{n^2}{\log_2 n}\right)$).

(B) is false. The running time of DFS is $O(n+m)$, and in this case m may be as large as n^2.

(C) is true. Since the degree is at most 7, there are at most $7n/2$ edges, so BFS takes time $O(n+m) = O(n)$, as $m = O(n)$.

(D) is true: We saw in class that an in-order traversal takes time $O(n)$.

(c) **(12 pt.)** Let $G = (V, E)$ be a directed acyclic graph, and suppose that $u, v \in V$ so that there is a directed edge from u to v. Which of the following must be true?

Note: Below, when we say "run DFS" or "run BFS," we mean on the whole graph. That is, if the graph is disconnected or not strongly connected, and there is nowhere for BFS/DFS to go but you haven't yet explored all the vertices, start again at an arbitrary vertex until you have reached them all.

☐ (A) If we were to run DFS on G, the finish time of u must be larger than the finish time for v.

☐ (B) Any run of BFS on G must visit u before visiting v.

☐ (C) u and v must be in the same strongly connected component of G.

☐ (D) If we were to run DFS on G, then v must be a descendant of u in the DFS tree.

☐ (E) None of the above.

SOLUTION:
A is true: we proved this in class when we were analyzing the topological sorting algorithm

B is false. For example, if we start BFS at v, it will visit v before u.

C is false. For example, if the graph is just $V = \{u, v\}$ and $E = \{(u, v)\}$, then there are two SCCs, $\{u\}$ and $\{v\}$.

D is false. For example, if we started DFS at v, then v would be the root and wouldn't be a descendant of anything. (It is true that if DFS discovers u before v, then v would be a descendant of u in the DFS tree).

(d) **(12 pt.)** Which of the following properties are associated with an algorithmic problem being *wicked*?

☐ (A) There is not one obvious way to mathematically formalize the problem.

☐ (B) The problem is NP-hard.

☐ (C) When testing your problem to see if it will work well, that test will impact real people.

☐ (D) The problem does not have a "correct" answer, only "better" or "worse" answers, depending on who you ask.

☐ (E) None of the above.

SOLUTION:
A, C, and D are true.

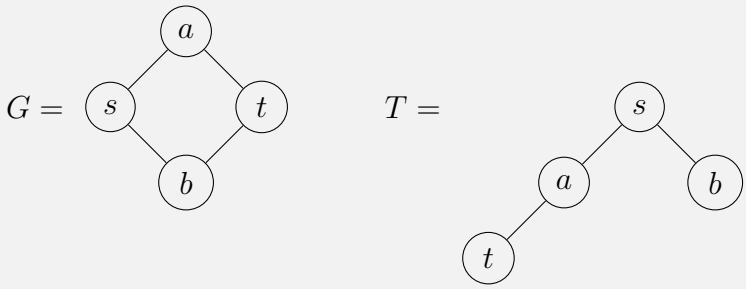
2. **(18 pt.)** [**True or False and Why?**] For each of the parts below, decide whether the statement is true or false. If it is true, give a short explanation (it doesn't need to be a formal proof). If it is false, give a counter-example.

   [**We are expecting:** *For each part, your answer (T or F), and either an explanation or a counter-example.*]

   (a) **(9 pt.)** Let $G = (V, E)$ be an undirected, unweighted graph. Say we run BFS on G, starting from a node s. Let $t \in V$ so that $t \neq s$.

   True or false: Every shortest path from s to t in G appears as a path in the BFS tree.

SOLUTION:
False. For example, if the graph G is as shown below, then T shown below would be a valid BFS tree. However, the shortest path $s \to b \to t$ does not appear as a path in T.



   (b) **(9 pt.)** Let $G = (V, E)$ be a directed, unweighted graph. Suppose that we run DFS on G, to obtain a DFS tree T. We say that an edge $(u, v) \in E$ is a *back edge* if u is a descendant of v in T.

   True or false: G has a directed cycle if and only if any DFS tree has a back edge.

   Practice Exam Note: This question might involve a bit more writing than we'd ask for on the real exam, but we hope that the conceptual difficulty is representative.

SOLUTION:
True. First, suppose that there is a back-edge (u, v) in any DFS tree T. By definition, u is a descendant of v in T, so there is a path from v to u. The edge (u, v) completes this path into a directed cycle.

On the other hand, suppose that there is a directed cycle in the graph, with vertices $v_0, v_1, \ldots, v_t \in V$ (so, $(v_0, v_1), (v_1, v_2), \ldots, (v_{t-1}, v_t), (v_t, v_0) \in E$). Consider any run of DFS, and suppose without loss of generality that $v_0 \in V$ is the first vertex in the cycle that DFS visits. (That is, v_0 is the vertex in the cycle with the smallest start time; if it is a different vertex instead of v_0, we can just re-label the

vertices). Then since there is a path from v_0 to v_t, and v_t hasn't been discovered yet when v_0 is discovered, v_t must appear as a descendant of v_0 in the DFS tree T. But then (v_t, v_0) is a back edge.

3. **(34 pt.) [How would you do it? (Short Answers)]** For each of the tasks below, say *briefly* how you would accomplish them. You may (and, hint, probably should) use any algorithm we have seen in class. If you slightly modify an algorithm we have seen in class, *very clearly* indicate what the modification would be.

   Below, all graphs have n vertices and m edges.

   [**We are expecting:** *For each of the tasks below, a short English description clearly describing an algorithm. Your description should be clear enough that a CS161 student who has been keeping up with the course (and the grader) can unambiguously understand your solution.* ***You do not need to write pseudo-code****, although you may if you want to.* ***You do not need to justify correctness or the running time.***]

   (a) **(10 pt.)** Suppose that $G = (V, E)$ is an unweighted connected graph. Give an algorithm that takes as input G and vertices $s, t, u \in V$, and outputs the fastest way to travel from s to t in the graph, stopping at u along the way. Your algorithm should run in time $O(n + m)$.

   SOLUTION:
   Run BFS to find a shortest path from s to u and then from u to t, and concatenate the two paths.

More parts on next page

(b) **(10 pt.)** Design a data structure that can hold n items and supports the following operations, each in time $O(\log n)$:

- INSERT, SEARCH, and DELETE, as discussed in class
- FINDMIN: Find the item in the array with the minimum key

SOLUTION:
Use a Red-Black Tree. We already saw how to do INSERT/SEARCH/DELETE in class. To implement FINDMIN, just go all the way to the left in the tree and return the element you get to when you can't go left any more.

(c) **(4 pt.)** *[Practice Exam Note: we'd possibly leave this part off a real exam to keep the length down, but it's a good practice problem.]* The same as the previous part (3b), but instead of FINDMIN, your data structure should support SELECT(k) in time $O(\log n)$. Here, SELECT(k) should return the item with the k'th smallest key.

Hint: In case it matters, to maintain the Red-Black tree property (the details of which we didn't cover in class), you only have to do rotations and some other local pointer manipulations. In particular, if you want to modify a RBTree so that each node keeps track of information like its depth (how far it is from the root); height (how far it is from the deepest leaf under it); or size (how many nodes live in the subtree under it), you can claim this without proof.

SOLUTION:
As with the previous problem, use an RBTree, but modify it so that each node knows how many elements live in the sub-tree beneath it (including itself). For a node v, call this $v.\texttt{count}$. The hint told us that we can do this, so we will.

Then when we are searching for the k'th smallest key, we start at the root r. Suppose that r's left child is A and right child is B. If $A.\texttt{count} = k-1$, then return r. If $A.\texttt{count} > k-1$, then recurse to find that k'th smallest key under A. If $A.\texttt{count} < k-1$, then recurse to find the ℓ'th smallest thing under B, where $\ell = k - A.\texttt{count} - 1$. (This looks a lot like the recursive algorithm we used in the k-SELECT divide and conquer algorithm from earlier in the quarter, except that we don't need to do FINDPIVOT, we can use the tree structure instead).

Another part on next page

(d) **(10 pt.)** Let $G = (V, E)$ be an unweighted, undirected, connected graph. Say that $v \in V$ is *disconnecting* if removing v (and all the edges that touch it) would disconnect G.

Suppose that your friend ran DFS on G, starting from v, and gave you a pointer to the root of the resulting DFS tree. In time $O(1)$, decide if v is disconnecting. Note that you have access to this DFS tree (including any meta-information left over from the DFS run), but not to G itself.

SOLUTION:

If v has at least two children in the DFS tree, then it is disconnecting. Otherwise it is not.

Not required: To see why this algorithm works, suppose that v is disconnecting. That means that there is some vertex x and some vertex y so that the only paths from x to y go through v. We claim that x and y must live in *different* sub-trees under v in the DFS graph. Indeed, if they lived in the same subtree, then there would be a path from x to y in that subtree that did not involve v. Thus, v must have at least two children in the DFS tree, one ancestor of x, and one ancestor of y.

On the other hand, suppose that v is *not* disconnecting. Then we claim that v can only have one child in the DFS tree. Suppose towards a contradiction that v has two children, x and y. We claim that the only paths from x to y must go through v. Indeed, suppose without loss of generality the DFS explored x before y. If there were a path from x to y that did not involve v, DFS would have gone from x, and then would have eventually gotten to y, before backtracking to v. So y would have been a descendent of x in the DFS tree, contradicting our assumption that x and y were both children of v. Thus v can have at most one child, as desired.

This is the end of the exam!

This is the end of the exam! You can use this page for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for, and make a note on the problem page itself.

This page is for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for, and make a note on the problem page itself.

This page is for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for, and make a note on the problem page itself.