CS161 Practice Final Exam

Do not turn this page until you are instructed to do so!

Instructions: Solve all questions to the best of your abilities. You may cite any result we have seen in lecture or the textbook without proof. You have **180 minutes** to complete this exam. You may use two two-sided sheets of notes that you have prepared yourself. You may not use any other notes, books, or online resources. There is one blank page at the end that you may tear off as scratch paper, and one blank page for extra work. Please write your name at the top of all pages.

Advice: If you get stuck on a problem, move on to the next one. Pay attention to how many points each problem is worth. Read the problems carefully.

The following is a statement of the Stanford University Honor Code:

1. *The Honor Code is an undertaking of the students, individually and collectively:*

   (1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

   (2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

2. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

3. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By signing your name below, you acknowledge that you have abided by the Stanford Honor Code while taking this exam.

Signature: _____

Name: _____

SUNetID: _____

Section	1 (multiple choice)	2 (short answer)	3 (alg. design)	4 (alg. analysis)	Total
Score					
Maximum	20	32	33	15	100

1 Multiple Choice (20 pts)

No explanation is required for the questions in Section 1. Please clearly mark your answers; if you must change an answer, either erase thoroughly or else make it **very** clear which answer you intend. **Ambiguous answers will be marked incorrect.**

1.1. **(2 pt.)** Below, assume that all functions $f(n)$ map positive integers to positive integers. Which of the following functions $f(n)$ are $O(n)$? Circle all that apply.

(A) $f(n) = 2n$

(B) $f(n) = \sqrt{n}$

(C) $f(n) = n\log(n) + 10$

(D) Any $f(n)$ that satisfies $f(n) = \Omega(n)$

(E) Any $f(n)$ that satisfies the recurrence $f(n) \leq 2 \cdot f(\lceil n/2 \rceil) + n$ for $n \geq 2$.

(F) Any $f(n)$ that satisfies the recurrence $f(n) \leq 3 \cdot f(\lceil n/2 \rceil) + n$ for $n \geq 2$.

(G) Any $f(n)$ that satisfies the recurrence $f(n) \leq f(\lceil n/2 \rceil) + f(\lceil n/20 \rceil) + n$ for $n \geq 20$.

1.2. **(2 pt.)** Suppose we run DFS on a graph G. Suppose that v and w are vertices in G, and in our run of DFS we assign start and finish times to these vertices. Which of the following are possible? Circle all that apply.

(A) v.start $\leq$ w.start $\leq$ w.finish $\leq$ v.finish

(B) w.start $\leq$ w.finish $\leq$ v.start $\leq$ v.finish

(C) v.start $\leq$ w.start $\leq$ v.finish $\leq$ w.finish

(D) w.start $\leq$ v.start $\leq$ w.finish $\leq$ v.finish

(E) w.start $\leq$ v.finish $\leq$ v.start $\leq$ w.finish

1.3. **(2 pt.)** Which of the following can RadixSort sort in time $O(n)$? Circle all that apply. (Assume that you can represent any integer in any basis in time $O(1)$).

(A) n positive integers of value at most 10

(B) n positive integers of value at most n

(C) n positive integers of value at most n^{10}

(D) n positive integers of value at most 10^n

(E) n positive integers of value at most n^n

1.4. **(2 pt.)** Suppose you have n dice, each with m sides. Let $M[x, m, n]$ denote the number of ways there are to roll a number x using your n dice. For example, if $m = 6$, $n = 2$, and $x = 7$, then there are six ways to roll 7 using two six-sided dice: $(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)$. Thus, $M[7, 6, 2] = 6$. You decide to write a dynamic programming algorithm to fill in the table M. Which of the following is the correct recursive structure? Circle exactly one. **(Note: don't worry about how you would actually implement the DP algorithm or about base cases, that's not part of this problem.)**

(A) $M[x, m, n] = M[x - 1, m, n - 1] + M[x - 1, m, n - 2] + 1$

(B) $M[x, m, n] = \sum_{i=1}^{m} M[x - i, m, n - 1]$

(C) $M[x, m, n] = \sum_{i=1}^{n} M[x, m, i]$

(D) $M[x, m, n] = M[x - 1, m, n - 1] + M[x - 1, m, n - 2] - 1$

1.5. **(8 pt.)** For each of the following quantities, fill in a single expression from the choices below that describes it. **Below, all graphs G have n vertices and m edges.**

1.5.1. The time it takes to search for an element in a red-black tree which is storing n items:

_____

1.5.2. The time it takes to deterministically sort n arbitrary comparable objects: _____

1.5.3. The expected number of items hashed into any given bucket, when n items are hashed into n buckets using a hash function h that is chosen uniformly at random from a universal hash family:

_____

1.5.4. The expected number of times you need to look at a random element from an array A of length n containing distinct integers until you see the maximum element of A:

_____

1.5.5. The time it takes to find an ordering $v_1, \ldots, v_n$ of the vertices in a directed acyclic graph $G = (V, E)$, so that for every directed edge $(v_i, v_j) \in E$, $i < j$:

_____

1.5.6. The number of edges in a minimum spanning tree in a connected undirected graph:

_____

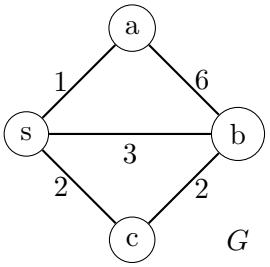
1.5.7. The worst-case running time of the Bellman-Ford algorithm: _____

1.5.8. The time it takes to determine if an unweighted undirected graph is bipartite: _____

Choices (which may be used more than once or not at all):

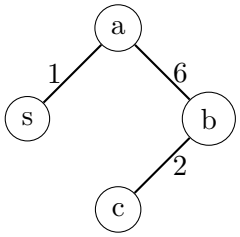
- $\Theta(n)$
- $\Theta(n \log(n))$
- $\Theta(n^2)$
- $\Theta(nm)$
- $\Theta(\log(n))$
- $O(1)$
- $\Theta(n + m)$

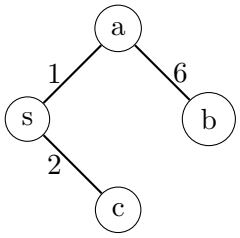
1.6. **(4 pt.)** Consider the undirected weighted graph G shown below. For each algorithm on the left, draw a single line connecting it to the sub-tree on the right naturally associated with the algorithm. A tree may be used more than once or not at all.

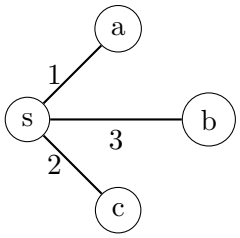


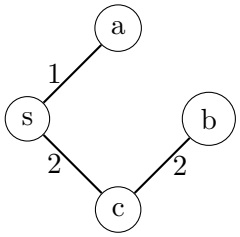
G

Dijkstra's algorithm, starting from s
BFS, starting from s, breaking ties in alphabetical order and ignoring edge weights
DFS, starting from s, breaking ties in alphabetical order and ignoring edge weights
Prim's algorithm, starting from s









2 Can it be done? (Short answers) (32 pts)

For each of the following tasks, either **explain briefly how you would accomplish it**, or else **explain why it cannot be done**. If you explain how to do it you do not need to justify why your answer is correct. You may use any algorithm or result we have seen in class as a black box.

Below, all graphs have n vertices and m edges.

The first two have been done for you to give an idea of the level of detail we are expecting. Note that it is possible to get full credit on this section without writing any pseudocode, although you may write pseudocode if you like.

2.1. **(0 pt.)** Find the maximum of an unsorted array of length n in time $O(n \log(n))$.

I would use MergeSort to sort the array, and then return the last element of the sorted array.

2.2. **(0 pt.)** Find the maximum of an unsorted array of length n in time $O(1)$.

This cannot be done, because since the maximum could be anywhere, we need to at least look at every element in the array, which takes time $\Omega(n)$.

2.3. **(6 pt.)** Given a weighted directed graph G with non-negative edge weights, and given vertices s and t, deterministically find a shortest path from s to t in time $O((m+n)\log(n))$.

2.4. **(6 pt.)** Given an undirected unweighted graph G with maximum degree[1] d and a vertex s, find all of the vertices v with distance at most 6 from s, in time $O(d^6)$.

2.5. **(4 pt.)** Search for an element in a sorted array, which contains n distinct arbitrary comparable items, in time $O(1)$.

2.6. **(6 pt.)** Given a directed weighted graph G, possibly with negative edge weights, decide whether or not the graph contains a negative cycle in time $O(n^3)$.

[1]Recall that the *degree* of a vertex is the number of edges leaving that vertex. The maximum degree of a graph is the maximum degree of any vertex in the graph.

Note: Problems 2.9 and 2.10 may be more difficult. You may wish to skip them and come back to them later.

2.7. **(4 pt.)** *(May be more difficult)* Suppose that you have access to a genie which can solve the all-pairs-shortest-path problem in directed acyclic graphs in time $O(n^2)$. That is, given a weighted DAG $G' = (V', E')$ the genie can return an $n \times n$ array containing the distances between any pair of vertices in V'.

Let $G = (V, E)$ be a weighted directed graph so that the largest strongly connected component in G has size 10. Give an algorithm which uses the genie and which can solve the all-pairs-shortest-path problem on G in time $O(n^2)$. (That is, it outputs an $n \times n$ array containing the distances between any two vertices in V).

2.8. **(4 pt.)** *(May be more difficult)* Let $G = (V, E)$ be a directed unweighted graph. Say that G is "kind-of-connected" if for every $u, v \in V$, either there is a path from u to v in G, or there is a path from v to u in G, or both. Decide if G is kind-of-connected in time $O(n + m)$.

3 Algorithm Design (33 pts)

3.1. **(11 pt.)** Suppose that A is an $n \times n$ array containing arbitrary positive integers. Design an algorithm that runs in time $O(n^2 \log(n))$ and finds the largest element that appears at least once in each row of A. For example, if

$$A = \begin{bmatrix} 4 & 1 & 2 & 8 \\ 2 & 5 & 4 & 1 \\ 3 & 4 & 6 & 3 \\ 4 & 7 & 4 & 2 \end{bmatrix},$$

then your algorithm should return 4.

[**We are expecting: Pseudocode, a high-level description of your algorithm, and a short justification of the running time.**]

3.2. **(11 pt.)** There are n final exams today at Stanford; exam i is scheduled to begin at time a_i and end at time b_i. Two exams which overlap cannot be administered in the same classroom; two exams i and j are defined to be *overlapping* if $[a_i, b_i] \cap [a_j, b_j] \neq \emptyset$ (including if $b_i = a_j$, so one starts exactly at the time that the other ends). Design a **greedy algorithm** which solves the following problem.

Input: Arrays A and B of length n so that $A[i] = a_i$ and $B[i] = b_i$.

Output: The smallest number of classrooms necessary to schedule all of the exams, and an optimal assignment of exams to classrooms.

For example: Suppose there are three exams, with start and finish times as given below:

i	1	2	3
a_i	12pm	4pm	2pm
b_i	3pm	6pm	5pm

Then the exams can be scheduled in two rooms; Exam 1 and Exam 2 can be scheduled in Room 1 and Exam 3 can be scheduled in Room 2.

Your algorithm should run in time $O(n \log(n) + nk)$, where k is the minimum number of classrooms needed.

[We are expecting: Pseudocode AND a short English description. You do not need to prove that your algorithm is correct or justify the running time.]

3.3. **(11 pt.)** Suppose that A is an $n \times n$ array that contains only zeros and ones. Your goal is to find the largest square in A that is all ones. That is, you want to find i, j and ℓ so that ℓ is as large as possible and so that the $\ell \times \ell$ subarray of A whose upper-right corner is (i, j) is entirely ones.

For example, if A were the matrix

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=4$	1	0	0	0	0
$i=3$	0	1	1	0	1
$i=2$	0	1	1	1	1
$i=1$	1	1	1	1	1
$i=0$	0	1	1	1	1

then you could return $i = 2, j = 4, \ell = 3$, corresponding the the box drawn above. (Here, the lower-left corner of the matrix is indexed as $(0, 0)$).

In the questions on the next two pages, you will design an algorithm to perform this task.

[Problems on next page]

3.3.1. **(5 pt.)** Define a function $F(x, y)$ to be the side length of the largest all-one square whose upper-right corner is (x, y).

In the example above, $F(2, 4) = 3$, while $F(0, 0) = 0$.

Give an equation[2] that expresses $F(x, y)$ in terms of $F(x-1, y), F(x, y-1), F(x-1, y-1)$ and $A[x, y]$, and explain why your equation is correct.

[We are expecting: An equation and an informal but convincing argument that it is correct.]

[Another part on next page]

[2]For example, a completely incorrect answer that has the correct type would be

$$F(x, y) = \max\{F(x, y-1), F(x-1, y)\} + F(x-1, y-1)^2.$$

3.3.2. **(6 pt.)** Give an algorithm to return the largest all-one square. Your algorithm should take as input an $n \times n$ array A, and should return i, j, ℓ as described above. Your algorithm should run in time $O(n^2)$.

[We are expecting: Pseudocode along with an English description of the idea of your algorithm, as well as a short justification of the running time.]

4 Algorithm Analysis (15 pts)

4.1. **(10 pt.)**

Consider the following algorithm for finding a minimum spanning tree in a connected, weighted, undirected graph $G = (V, E)$.

```
def newMST(G):
     while there is a cycle in G:
          let C be any cycle in G
          remove the largest-weight edge from C
     return G
```

That is, while the algorithm can find a cycle in G, it deletes the edge with the largest weight in that cycle. When it can no longer find a cycle, then it returns whatever is left.

4.1.1. **(5 pt.)** Use the following lemma to write a proof by induction that `newMST` is correct: that is, that it always returns an MST of G. **You do not have to prove the lemma (yet).**

> **Lemma:** Let C be any cycle in G, and let $\{u, v\}$ be the edge in that cycle with the largest weight. Then there exists an MST of G that does *not* include edge $\{u, v\}$.

[We are expecting: Your inductive hypothesis, base case, and conclusion, and a description of how the lemma can establish the inductive step.]

[More space and more problems on next page]

(More space for Problem **??**).

4.1.2. **(5 pt.)** Prove the lemma.

4.2. **(5 pt.)** Let A be an array of n items from a universe $\mathcal{U}$, and suppose that $\mathcal{H}$ is a universal hash family so that the elements of $\mathcal{H}$ are functions $h : \mathcal{U} \to \{0, \ldots, b-1\}$, where b will be specified below.

Your goal is to decide if there are any repeated elements in A, and your friend comes up with the following randomized algorithm.

Algorithm 1: `isThereARepeat`(A)

Choose $h \in \mathcal{H}$ uniformly at random.
Initialize an array B of length b to all zeros.
for $i \in \{0, \ldots, n-1\}$ **do**
  if $B[h(A[i])] == 1$ **then**
    return `True`
  $B[h(A[i])] \leftarrow 1$
return `False`

The algorithm chooses a random hash function $h \in \mathcal{H}$, and hashes all of the elements of A. If there is ever a collision, the algorithm returns `True`, meaning that it guesses that there was a repeated element. Otherwise, it returns `False`, meaning that there was no repeated element.

Suppose that $b = 10n^2$. Prove that the algorithm is correct with probability at least 9/10. More precisely, prove that (a) if A has a repeated element, then `isThereARepeat`(A) always returns `True`; and (b) if A does not have a repeated element, then `isThereARepeat`(A) returns `False` with probability at least 9/10.

This is the end!

This page intentionally blank for extra space for any question.
Please indicate in the relevant problem if you have work here that you want graded, and label your work clearly.

This page intentionally blank for extra space for any question.
Please indicate in the relevant problem if you have work here that you want graded, and label your work clearly.

This page is for **scratch space**. You may tear off this page. Nothing on this page will be graded.

This page is for **scratch space**. You may tear off this page. Nothing on this page will be graded.