Final Exam

- Do not open this exam booklet until you are directed to do so. Read all the instructions on this page.
- When the exam begins, write your name on every page of this exam booklet.
- This exam contains 12 problems, some with multiple parts. You have 180 minutes to earn 180 points.
- This exam booklet contains 24 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your exam at the end of the exam period.
- This exam is closed book. You may use three $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader		Problem	Parts	Points	Grade	Grader
1	1	10				7	2	10		
2	10	40				8	2	10		
3	1	10				9	1	10		
4	2	10				10	3	15		
5	1	10				11	4	20		
6	2	15				12	4	20		
						Total		180		

Name: _____

Circle your recitation time:
Hueihan Jhuang: **(10AM) (11AM)** Victor Costan **(2PM) (3PM)**

Problem 1. Asymptotics [10 points]

For each pair of functions $f(n)$ and $g(n)$ in the table below, write O, Ω, or Θ in the appropriate space, depending on whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. If there is more than one relation between $f(n)$ and $g(n)$, write only the strongest one. The first line is a demo solution.

We use $\lg$ to denote the base-2 logarithm.

	n	$n \lg n$	n^2
$n \lg^2 n$	Ω	Ω	O
$2^{\lg^2 n}$			
$\lg(n!)$			
$n^{\lg 3}$			

Problem 2. True or False [40 points] (10 parts)

Decide whether these statements are **True** or **False**. You must briefly justify all your answers to receive full credit.

(a) An algorithm whose running time satisfies the recurrence $P(n) = 1024\,P(n/2) + O(n^{100})$ is asymptotically faster than an algorithm whose running time satisfies the recurrence $E(n) = 2\,E(n - 1024) + O(1)$.

True **False**

Explain:

(b) An algorithm whose running time satisfies the recurrence $A(n) = 4\,A(n/2) + O(1)$ is asymptotically faster than an algorithm whose running time satisfies the recurrence $B(n) = 2\,B(n/4) + O(1)$.

True **False**

Explain:

(c) Radix sort works in linear time **only if** the elements to sort are integers in the range $\{0, 1, \ldots, c\,n\}$ for some $c = O(1)$.

True **False**

Explain:

(d) Given an undirected graph, it can be tested to determine whether or not it is a tree in $O(V + E)$ time. A *tree* is a connected graph without any cycles.

True **False**

Explain:

(e) The Bellman-Ford algorithm applies to instances of the single-source shortest path problem which do not have a negative-weight directed cycle, but it does not detect the existence of a negative-weight directed cycle if there is one.

True **False**

Explain:

(f) The topological sort of an arbitrary directed acyclic graph $G = (V, E)$ can be computed in linear time.

True **False**

Explain:

(g) We know of an algorithm to detect negative-weight cycles in an arbitrary directed graph in $O(V + E)$ time.

True **False**

Explain:

(h) We know of an algorithm for the single source shortest path problem on an arbitrary graph with no negative-weights that works in $O(V + E)$ time.

True **False**

Explain:

(i) To delete the i^{th} node in a min heap, you can exchange the last node with the i^{th} node, then do the min-heapify on the i^{th} node, and then shrink the heap size to be one less the original size.

True **False**

Explain:

(j) Generalizing Karatsuba's divide and conquer algorithm, by breaking each multiplicand into 3 parts and doing 5 multiplications improves the asymptotic running time.

True **False**

Explain:

Problem 3. Set Union [10 points]

Give an efficient algorithm to compute the union $A \cup B$ of two sets A and B of total size $|A|+|B| = n$. Assume that sets are represented by arrays (Python lists) that store distinct elements in an arbitrary order. In computing the union, the algorithm must remove any duplicate elements that appear in both A and B.

For full credit, your algorithm should run in $O(n)$ time. For partial credit, give an $O(n \lg n)$-time algorithm.

Problem 4. Balanced Trees [10 points]

In the definition of an AVL tree we required that the height of each left subtree be within one of the height of the corresponding right subtree. This guaranteed that the worst-case search time was $O(\log n)$, where n is the number of nodes in the tree. Which of the following requirements would also provide the same guarantee?

(a) The number of nodes in each left subtree is within a factor of 2 of the number of nodes in the corresponding right subtree. Also, a node is allowed to have only one child if that child has no children.

This tree has worst case height $O(\lg n)$.

True **False**

Explain:

(b) The number of leaves (nodes with no children) in each left subtree is within one of the number of leaves in the corresponding right subtree.

This tree has worst case height $O(\lg n)$.

True **False**

Explain:

Problem 5. Height Balanced Trees [10 points]

We define the height of a node in a binary tree as the number of nodes in the longest path from the node to a descendant leaf. Thus the height of a node with no children is 1, and the height of any other node is 1 plus the larger of the heights of its left and right children.

We define height balanced trees as follows;

- each node has a "height" field containing its height,
- at any node, the height of its right child differs by at most one from the height of its left child.

Finally we define $\texttt{Fib}(i)$ as follows,

$$\texttt{Fib}(0) = 1$$

$$\texttt{Fib}(1) = 1$$

$$\texttt{Fib}(i) = \texttt{Fib}(i-1) + \texttt{Fib}(i-2), \text{ for } i \geq 2.$$

You may use without proof that $\texttt{Fib}(n) \geq 1.6^n$ for large n.

Prove that there are at least $\texttt{Fib}(h)$ nodes in a height balanced tree of height h, for all $h \geq 1$.

Problem 6. Maintaining Medians [15 points]

Your latest and foolproof (really this time) gambling strategy is to bet on the median option among your choices. That is, if you have n distinct choices whose sorted order is $c[1] < c[2] < \cdots < c[n]$, then you bet on choice $c[\lfloor (n+1)/2 \rfloor]$. As the day goes by, new choices appear and old choices disappear; each time, you sort your current choices and bet on the median. Quickly you grow tired of sorting. You decide to build a data structure that keeps track of the median as your choices come and go. Specifically, your data structure stores the number n of choices, the current median m, and two AVL trees S and T, where S stores all choices less than m and T stores all choices greater than m.

(a) Explain how to add a new choice c_{new} to the data structure, and restore the invariants that (1) m is the median of all current choices; (2) S stores all choices less than m; and (3) T stores all choices greater than m. Analyze the running time of your algorithm.

(b) Explain how to remove an existing choice c_{old} from the data structure, and restore invariants (1–3) above. Analyze the running time of your algorithm.

Problem 7. Hashing [10 points]

Suppose that we have a hash table with $2n$ slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Assume simple uniform hashing, i.e., each key is equally likely to be hashed into each slot.

(a) What is the expected number of elements that hash into slot i?

(b) What is the probability that exactly k keys hash into slot i?

Problem 8. d**-max-heap** [10 points]

A ***d-max-heap*** is like an ordinary binary max-heap, except that nodes have d children instead of 2.

(a) Describe how a d-max-heap can be represented in an array $A[1 \ldots n]$. In particular, for the internal (non-leaf) node of the d-max-heap stored in any location $A[i]$, which positions in A hold its child nodes?

(b) Define the height of the heap to be the number of nodes on the longest path from the root to a leaf.

In terms of n and d, what is the height of a d-max-heap of n elements?

Problem 9. Firehose Optimization [10 points]

You have decided to apply your algorithmic know-how to the practical problem of getting a degree at MIT. You have just snarfed the course catalog from WebSIS. Assume there no cycles in course prerequisites. You produce a directed graph $G = (V, E)$ with two types of vertices $V = C \cup D$: regular *class vertices* $c \in C$ and special *degree vertices* $d \in D$. The graph has a directed edge $e = (u, v)$ whenever a class $u \in C$ is a prerequisite for $v \in V$ (either a class or a degree). For each class $c \in C$, you've computed your *desire* $w(c) \in \mathbb{R}$ for taking the class, based on interest, difficulty, etc. (Desires can be negative.)

Give an $O(V + E)$-time algorithm to find the most desirable degree, that is, to find a degree $d \in D$ that maximizes the sum of the desires of the classes you must take in order to complete the degree: $\sum\{w(c) : \text{path } c \rightsquigarrow d\}$. (For partial credit, give a slower algorithm.)

Problem 10. Histogram Hysterics [15 points]

Sometime in the future, you become a TA for 6.006. You have been assigned the job of maintaining the grade spreadsheet for the class. By the end of the semester, you have a list g of final grades for the n students, sorted by grade: $g[0] < g[1] < \cdots < g[n-1]$. In an attempt to draw various beautiful histograms of these grades, the (rather odd) professors now ask you a barrage of questions of the form "what is the sum of grades $g[i:j]$, i.e., $g[i] + g[i+1] + \cdots + g[j-1]$?" for various pairs (i, j). (Dividing this sum by $j - i$ then gives an average.)

To save you work computing summations, you decide to compute some of the sums $g[i:j]$ ahead of time and store them in a data structure. Unfortunately, your memory is large enough to store only $\Theta(n)$ such sums. Once these sums have been computed, can you answer each query by the professors in $O(1)$ time? If not, give the fastest solution you can.

(a) Which sums $g[i:j]$ should you compute ahead of time?

(b) In what data structure should you store these sums?

(c) How do you then compute a professors' query for an arbitrary sum $g[i : j]$, and how long does this take?

Problem 11. Wonderland [20 points]

You have just taken a job at Wonderland (at the end of the Blue Line) as an amusement-ride operator. Passengers can enter the ride provided it is not currently running. Whenever you decide, you can run the ride for a fixed duration d (during which no passengers can enter the ride). This action brings joy to the passengers, causing them to exit the ride and pay you d/t_i dollars where t_i is the amount of time passenger i spent between arriving and exiting the ride. Thus, if you start the ride as soon as a passenger arrives, then $t_i = d$, so you get \$1.00 from that passenger. But if you wait d units of time to accumulate more passengers before starting the ride, then $t_i = 2d$, so you only get \$0.50 from that passenger.

Every day feels the same, so you can predict the arrival times $a_0, a_1, \ldots, a_{n-1}$ of the n passengers that you will see. As passenger i arrives, you must decide whether to start the ride (if it is not already running). If you start the ride at time a_j, then you receive $d/(d + a_j - a_i)$ dollars from customers $i \leq j$ that have not yet ridden, and you can next start the ride at times $a_k \geq a_j + d$. Your goal is to maximize the total amount of money you make using dynamic programming.

(a) Clearly state the set of subproblems that you will use to solve this problem.

(b) Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.

(c) Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem.

(d) Write the solution to the original problem in terms of solutions to your subproblems.

Problem 12. Dance Dance Evolution [20 points]

You are training for the upcoming Dance Dance Evolution competition and decide to apply your skills in dynamic programming to find optimal strategies for playing each song.

A simplified version of the problem can be modeled as follows. The song specifies a sequence of "moves" that you must make. Each *move* is one of four buttons in the set $B = \{\boxed{\uparrow}, \boxed{\downarrow}, \boxed{\leftarrow}, \boxed{\rightarrow}\}$ that you must press with one of your feet. An example of a song is

$$\boxed{\leftarrow}, \boxed{\rightarrow}, \boxed{\uparrow}, \boxed{\downarrow}, \boxed{\rightarrow}, \boxed{\leftarrow}, \boxed{\downarrow}, \boxed{\uparrow}.$$

You have two feet. At any time, each foot is on one of the four buttons; thus, the current state of your feet can be specified by an ordered pair (L, R) where $L \in B$ denotes the button beneath your left foot and $R \in B$ denotes the button beneath your right foot.

One foot at a time: When you reach a move $M \in B$ in the song, you must put one of your feet on that button, transitioning to state (M, R) or (L, M). Note that you can change only one of your feet per move. If you already have a foot on the correct button, then you do not need to change any feet (though you are allowed to change your other foot).

Forbidden states: You are also given a list of forbidden states $\mathcal{F}$, which you are never allowed to be in. $\mathcal{F}$ might include states where both feet are on the same square, or states where you would end up facing backwards.

Your goal is to develop a polynomial-time algorithm that, given an n-move song $M_1, M_2, \ldots, M_n$, finds an initial state (L_0, R_0) and a valid sequence of transitions $(L_i, R_i) \to (L_{i+1}, R_{i+1}) \notin \mathcal{F}$, for $0 \leq i < n$, where $M_{i+1} \in \{L_{i+1}, R_{i+1}\}$ and either $L_i = L_{i+1}$ or $R_i = R_{i+1}$,

(a) Clearly state the set of subproblems that you will use to solve this problem.

(b) Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.

(c) Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem.

(d) Write the solution to the original problem in terms of solutions to your subproblems.

SCRATCH PAPER

SCRATCH PAPER