Introduction to Algorithms December 19, 2001
Massachusetts Institute of Technology 6.046J/18.410J
Singapore-MIT Alliance SMA5503
Professors Erik Demaine, Lee Wee Sun, and Charles E. Leiserson Final Exam

Final Exam

- Do not open this exam booklet until you are directed to do so. Place all your belongings on the floor except for your exam materials.
- This exam contains 6 problems. There are 19 pages to the exam plus 2 scratch pages. You have 180 minutes to earn 180 points.
- This exam is closed book, but you may use two handwritten letter-size crib sheets.
- When the exam begins, write your name on every page in this exam booklet.
- Write your solutions in the space provided. If you need more space, complete your answer on the back of the sheet containing the problem, and indicate in the space provided that your answer is continued. Do not put part of the answer to one problem on the back of the sheet for another problem.
- Do not spend too much time on any 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 and efficiency of your answers, but also on your clarity. Be neat.

Problem	Title	Parts	Points	Score
1	True or False	10	60	
2	Divide and Conquer	1	10	
3	Competitive Analysis	2	20	
4	Flow Networks	1	10	
5	Short Problems	4	40	
6	Ruling Them All	4	40	
Total			180	

Name: ____________________

MIT students: Please circle your section:

10:00	11:00	11:00	12:00	12:00	1:00	1:00	2:00	2:00	3:00
Jen	Jen	Chong	Chong	Rachel	Rachel	Bob	Bob	George	George

Problem 1. [60 points] (10 parts) **True or False**

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

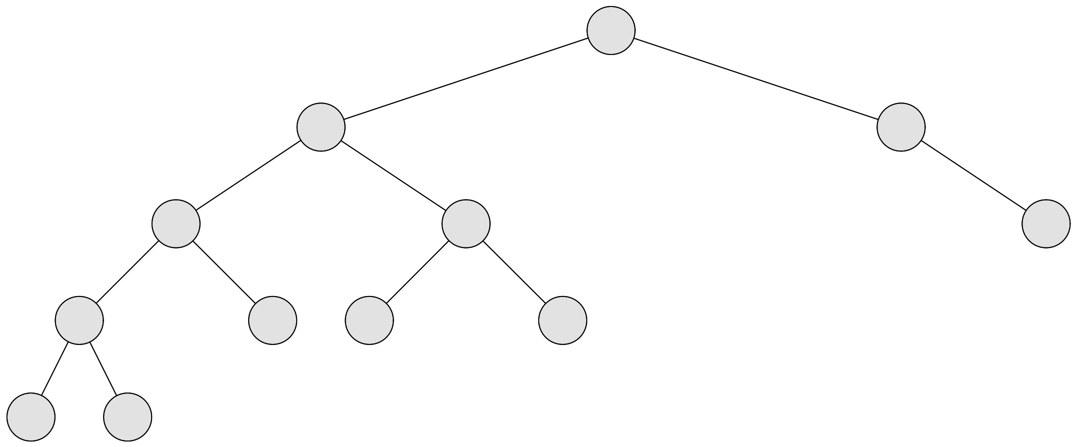
(a) **T F** On an input array of 6 numbers, a comparison sort requires at least 10 comparisons in the worst case.

(b) **T F** Any n-node tree that satisfies both the min-heap property and the binary-search-tree property on the same keys has height $\Theta(n)$.

(c) **T F** The number of distinct elements in an array of size n can be determined in $O(n)$ expected time.

(d) **T F** Suppose that you are given a parallel-processing subroutine that can find the median of n numbers in $O(\sqrt{n})$ time. Then, you can sort n numbers in $O(n)$ time.

(e) T F The nodes of the following tree can be colored such that it is a red-black tree. (Leaves (NIL's) are not shown.)



(f) **T F** For a binary min-heap, a potential function can be found such that the amortized cost of an INSERT operation on a heap of n elements is $O(\lg n)$ and the amortized cost of an EXTRACT-MIN operation is $O(1)$.

(g) **T F** Whether an undirected graph $G = (V, E)$ contains a cycle can be detected in $O(V)$ time in the worst case.

(h) **T F** Any digraph $G = (V, E)$ contains at most $O(VE)$ distinct simple cycles. (Two cycles are ***different*** if they differ in at least one edge.)

(i) **T F** The single-source shortest-paths problem on a digraph $G = (V, E)$ can be solved in $O((V + E) \lg \lg V)$ time if the edge weights are integers in the range from 1 to $|V|^2$.

(j) **T F** In a digraph $G = (V, E)$ where all edge weights are equal to 1, the shortest path from every vertex $u \in V$ to a given destination vertex v can be found in $O(V+E)$ time.

Problem 2. [10 points] **Divide and Conquer**

The asymptotic performance of a particular divide-and-conquer program can be described by the recurrence $T(n) = 5T(n/2) + \Theta(n^2)$. Because the structure of the program and its asymptotic performance cannot be changed, the programmer is constrained to improve the constant factors in its running time. Where should the programmer prioritize recoding effort, and why? (For example, improving the code for the base of the recursion, the divide step, the combine step, etc.)

Problem 3. [20 points] (2 parts) **Competitive Analysis**

You are just about to take up the sport of algolagnia, and you must decide whether to rent or buy equipment. Algolagnia is a dangerous sport, however, and you risk permanent injury on each outing, in which case you will never play the sport again.

You are faced with a difficult choice. You can rent equipment for r dollars per outing, or you can buy equipment for $b \gg r$ dollars once, and then each outing is free. If you rent and continue to play the sport for a long time, you may find yourself squandering a lot of money. If you buy and injure yourself shortly thereafter, however, you may also find yourself wasting a lot of money.

You examine the following strategy:

Rent equipment for b/r outings, and then buy.

(You may assume that r evenly divides b.)

(a) Prove that this strategy is $O(1)$-competitive. Give a good bound on the constant hidden by the big-O.

Not satisfied with the competitive ratio provided by this deterministic strategy, you decide to examine a randomized strategy as well:

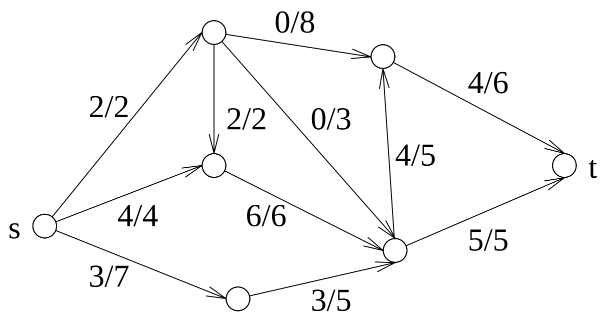
> Rent equipment for $b/2r$ outings, and then flip a coin which is biased to come up heads with probability p and tails with probability $1-p$. If the coin comes up heads, you buy equipment immediately. If the coin comes up tails, you rent equipment for another $b/2r$ outings and then buy.

(You may assume that $2r$ evenly divides b.)

(b) Pick a value for p that optimizes the expected competitive ratio of this strategy, and give a good bound on the expected competitive ratio. (For partial credit, simply find a value for p that beats the good bound asked for in part (a).)

Problem 4. [10 points] **Flow Networks**

Draw the residual graph for the following flow network, and find the augmenting path that has maximum capacity.



Problem 5. [40 points] (4 parts) **Short Problems**

(a) Explain the difference between ***universal*** hashing and ***perfect*** hashing. When would you use each?

(b) Suppose that in an edge-weighted, undirected graph $G = (V, E)$, the edge (u, v) is the unique edge incident to u having the least weight, that is, every other edge incident to u has strictly greater weight. Show that (u, v) belongs to every minimum spanning tree of G.

(c) The Bellman-Ford algorithm run on a digraph $G = (V, E)$ sets $\pi[v] \leftarrow u$ whenever the relaxation of an edge $(u, v) \in E$ is ***successful***, that is, whenever the value of $d[v]$ changes. Show that if the graph $G' = (V, E')$, where $E' = \{(\pi[v], v)\}$, contains a cycle, then G contains a negative-weight cycle. (For partial credit, show that G contains a nonpositive-weight cycle.)

(d) Let $G = (V, E)$ be a flow network, and let f_1 and f_2 be flows on G. Prove that the flow f given by

$$f(u, v) = \frac{f_1(u, v) + f_2(u, v)}{2}$$

is also a flow on G.

Problem 6. [40 points] (4 parts) **Ruling Them All**

You've recently become the Dark Lord of Mordor. You control n balrogs $b_1, b_2, \ldots, b_n$, each with many jobs to get done, and m orcs $o_1, o_2, \ldots, o_m$, each willing to work for exactly one balrog. Each orc o_j has achieved a particular score OQE_j on the Orc Qualifying Exam. Each balrog b_i has many identical jobs requiring an OQE score of at least $\mathit{min\text{-}OQE}_i$ and at most $\mathit{max\text{-}OQE}_i$. In payment for the job, each balrog b_i offers a fixed number food_i of grams of hobbit flesh. Each orc o_j requires at least $\mathit{min\text{-}food}_j$ grams of hobbit and at most $\mathit{max\text{-}food}_j$ grams of hobbit.

Your goal is to assign (whenever possible) a balrog to each orc. The assignment should be ***suitable*** in the sense that each balrog must be willing to hire each assigned orc (appropriate OQE score), and each orc must be willing to work for the assigned balrog (appropriate food). Assume that each balrog has an unlimited number of identical jobs.

(a) Describe how to model this problem in terms of a collection of line segments in the plane.

You immediately think up a sweepline algorithm. The algorithm maintains a dynamic set of “current” orcs, keyed by their OQE scores. Initially no orcs are current. The algorithm passes through all *food*$_i$’s, *min-food*$_j$’s, and *max-food*$_j$’s in increasing order:

1. Whenever the algorithm encounters the minimum acceptable food amount *min-food*$_j$ of a new orc o_j, it adds that orc to the dynamic set.
2. Whenever the algorithm encounters the maximum acceptable food amount *max-food*$_j$ of a (current) orc o_j, it removes that orc from the dynamic set.
3. Whenever the algorithm encounters the food amount *food*$_i$ offered by a balrog b_i, it searches through the dynamic set for all orcs with OQE scores between *min-OQE*$_i$ and *max-OQE*$_i$, assigns those orcs to balrog b_i, and removes those orcs from the dynamic set.

(b) Explain how each execution of Step 1 and Step 2 can each be implemented in $O(\lg m)$ time. Explain how each execution of Step 3 can be implemented in $O(k + \lg m)$ time, where k is the number of orcs assigned jobs during that step. Be explicit about what data structure you have chosen to implement the dynamic set.

(c) Argue that the total running time of the algorithm is $O((m + n)\lg(m + n))$. (For partial credit, argue a weaker bound.)

(d) Consider the more realistic situation in which each balrog b_i has a limited number p_i of job positions available. Thus, at most p_i different orcs can be assigned to balrog b_i. Briefly describe an algorithm to assign orcs to suitable balrogs in order to maximize the total number of orcs assigned. Your algorithm should run in polynomial time ($O(n^c)$ time for some constant $c \geq 0$), but you need not analyze your algorithm.