Practice Final Exam Solutions

Problem Final-1. [50 points] (9 parts)

In each part of this problem, circle *all* of the choices that correctly answer the question or complete the statement. Cross out the remaining (incorrect) choices, and in the space provided, briefly explain why the item is wrong. (You need not explain correct answers.) If you circle or cross out incorrectly, you will be penalized, so leave the choice blank unless you are reasonably sure.

(a) Which of the following algorithms discussed in 6.046 are greedy?

1. Prim's algorithm for finding a minimum spanning tree
2. finding an optimal file merging pattern
3. ~~finding a longest common subsequence of two strings~~
4. Dijkstra's algorithm for solving the single-source shortest paths problem

3. Finding a longest common subsequence of two strings uses a dynamic programming approach.

(b) An adversary cannot elicit the worst-case behavior of:

1. ~~ordinary quicksort.~~
2. a hash table where universal hashing is used.
3. ~~a self-organizing list where the move-to-front (MTF) heuristic is used.~~
4. RANDOMIZED-SELECT for finding the kth smallest element of an array.

1. An adversary can beat a deterministic partitioning scheme with a bad input, such that all elements fall to one side of the pivot.
3. An adversary can always access the tail element of the list, regardless of the online heuristic.

(c) Which of the following can be performed in worst-case linear time in the input size?

1. building a binary heap from an unsorted list of numbers
2. determining if a graph is bipartite
3. walking a red-black tree inorder
4. solving the single-source shortest paths problem on an acyclic directed graph

(d) Which of the following statements about trees are correct?

1. Given a set S of n real keys chosen at random from a uniform distribution over $[a, b)$, a binary search tree can be constructed on S in $O(n)$ expected time.
2. In the worst case, a red-black tree insertion requires $O(1)$ rotations.
3. Given a connected, weighted, undirected graph G in which the edge with minimum weight is unique, that edge belongs to every minimum spanning tree of G.

~~4.~~ Deleting a node from a binary search tree on n nodes takes $O(\lg n)$ time in the worst case.

4. Deletion on an ordinary binary search tree of height h takes $\Theta(h)$ time in worst case, and h can be $\Omega(n)$ if the tree with n nodes is unbalanced.

(e) Which of the following algorithms discussed in 6.046 employ dynamic programming?

~~1.~~ Kruskal's algorithm for finding a minimum spanning tree

2. the Floyd-Warshall algorithm for solving the all-pairs shortest paths problem
3. optimal typesetting, where the line penalty is the cube of the number of extra spaces at the end of a line

~~4.~~ the Bellman-Ford algorithm for solving the single-source shortest paths problem

1. Kruskal's algorithm uses a greedy strategy.

4. The Bellman-Ford algorithm simply uses repeated edge relaxation.

(f) Which of the following correctly match a theoretical result from 6.046 with an important technique used in a proof of the result?

1. correctness of HEAPIFY ... induction on subtree size

~~2.~~ 4-competitiveness of the move-to-front (MTF) heuristic for self-organizing lists ... cut-and-paste argument

3. correctness of Dijkstra's algorithm for solving the single-source shortest paths problem ... well ordering and proof by contradiction
4. expected height of a randomly built binary search tree ... indicator random variables and Chernoff bound

2. This result uses an amortized or "potential-function" argument in its proof.

(g) On inputs for which both are valid, in the worst case:

~~1.~~ breadth-first search asymptotically beats depth-first search.

~~2.~~ insertion into an AVL tree asymptotically beats insertion into a 2-3-4 tree.

3. Dijkstra's algorithm asymptotically beats the Bellman-Ford algorithm at solving the single-source shortest paths problem.

~~4.~~ shellsort with the increment sequence $\{2^i 3^j\}$ has the same asymptotic performance as mergesort.

1. Both breadth-first and depth-first search are $O(V + E)$.
2. Both AVL trees and 2-3-4-trees are balanced, so insertion take $O(\lg n)$ time.
4. Mergesort, a $\Theta(n \lg n)$ algorithm, beats shellsort, a $\Theta(n \lg^2 n)$ algorithm.

Problem Final-2. [25 points] (5 parts)

In parts (a)–(d), give asymptotically tight upper (big O) bounds for $T(n)$ in each recurrence. Briefly justify your answers.

(a) $T(n) = 4T(n/2) + n^2$.

By case 2 of Master Theorem: $T(n) = \Theta(n^2 \lg n)$

(b) $T(n) = T(n/2) + n$.

By case 3 of Master Theorem: $T(n) = \Theta(n)$. *Note:* $f(n) = n$ *is regular.*

(c) $T(n) = 3T(n/3) + \lg n$.

By case 1 of Master Theorem: $T(n) = \Theta(n)$

(d) $T(n) = 2T(n/2) + \lg(n!)$.

By case 3 of Master Theorem: $T(n) = \Theta(n \lg^2 n)$. *Note:* $\Theta(\lg(n!)) = \Theta(n \lg n)$

(e) Use the substitution method to prove that the recurrence

$$T(n) = 2T(n/2) + n$$

can be bounded below by $T(n) = \Omega(n \lg n)$.

Assume $T(k) \geq c\, k \lg k$ *for* $k < n$.

$$\begin{aligned} T(n) &= 2\,T(n/2) + n \\ &\geq c\, n \lg(n/2) + n \\ &= c\, n \lg n - c\, n + n \\ &\geq c\, n \lg n \quad \textit{if } c \leq 1\,. \end{aligned}$$

The initial conditions are satisfiable, because $T(2) \geq 2c$ *and* $T(3) \geq 3c \lg 3$ *, which can be achieved by making* c *sufficiently close to* 0.

Problem Final-3. [25 points]

Use a potential-function argument to show that any sequence of insertions and deletions on a red-black tree can be performed in $O(\lg n)$ amortized time per insertion and $O(1)$ amortized time per deletion. (For substantial partial credit, use an aggregate or accounting argument.)

We define the potential function φ of a red-black tree T to be

$$\varphi(T) = k \sum_{i=1}^{n} \lg i \,,$$

where n is the number of nodes in T and $k > 0$ is a constant.

The initial tree T_0 is an empty red-black tree and we have $\varphi(T_0) = 0$. For any tree T obtained applying insertions and deletions to T_0 the potential function is $\varphi(T) \geq 0$. Hence φ is a well-defined potential function.

Next we compute the amortized cost

$$\hat{c} = c + \varphi(T_{\text{after}}) - \varphi(T_{\text{before}})$$

wherere $c =$ actual cost of the operation, $T_{\text{after}} =$ tree after the operation, $T_{\text{before}} =$ tree before the operation.

Let n be the number of nodes in T before the operation.

Amortized cost for insertion (actual cost $c \leq a_1 \lg n + a_2$):

$$\hat{c}_{\text{INSERTION}} = a_1 \lg n + a_2 + k \sum_{i=1}^{n+1} \lg i - k \sum_{i=1}^{n} \lg i = (a_1 + k)\ \lg(n+1) + a_2$$

Amortized cost for deletion (actual cost $c \leq b_1 \lg n + b_2$):

$$\hat{c}_{\text{DELETE}} = b_1 \lg n + b_2 + k \sum_{i=1}^{n-1} \lg i - k \sum_{i=1}^{n} \lg i = (b_1 - k)\ \lg n + b_2$$

By choosing $k = b_1$ we get: $\hat{c}_{\text{DELETE}} = O(1)$ and $\hat{c}_{\text{INSERTION}} = O(\lg n)$.

Problem Final-4. [25 points]

Bitter about his defeat in the presidential election, Rob Roll decides to hire a seedy photographer to trail Will Clintwood. (The names have been changed to protect the guilty.) The photographer stealthily takes pictures of Clintwood, and he marks each picture with the time t it was taken. Roll tells the photographer to mark especially scandalous pictures with a big, red X, because these pictures will be used in future negative advertisements. Roll requires a data structure that contains the Clintwood pictures and supports the following operations:

- INSERT(x)—Inserts picture x into the data structure. Picture x has an integer field *time*$[x]$ and a boolean field *scandalous*$[x]$.
- DELETE(x)—Deletes picture x from the data structure.
- NEXT-PICTURE(x)—Returns the picture that was taken immediately after picture x.
- SCANDAL!(t)—Returns the first scandalous picture that was taken after time t.

Describe an efficient implementation of this data structure. Show how to perform these operations, and analyze their running times. Be succinct.

To allow us to implement these operations efficiently, we'll use a red-black tree representation keying photographs by time, and augmenting each node x with these additional fields: scandalous?$[x]$ and max-scand-time$[x]$. The field max-scand-time$[x]$ stores the time when the latest scandalous picture in the subtree rooted at x was taken. It contains either an integer time, or NIL *if there are no scandalous pictures in the subtree. We implement the operations (all in $O(\lg n)$ running time) on this data structure as follows:*

INSERT(x)*: Place x in the tree, keyed by time$[x]$, just like in a regular red-black tree. Update max-scand-time fields if necessary after rotations. This can be done in $O(1)$ time.*

DELETE(x)*: Just like* INSERT(x)*; do what's normally done in a red-black tree, taking care to fix max-scand-time fields after deleting node x. When node n is encountered where max-scand-time$[n]$ = time$[x]$ and scandalous?$[x]$ = * TRUE*, look at the node's children and set max-scand-time$[n] \leftarrow$* $\max($*max-scand-time*$[$*right*$[n]]$, *max-scand-time*$[$*left*$[n]])$.

NEXT-PICTURE(x)*: Just like* SUCCESSOR(x) *in a red-black tree.*

SCANDAL!(t)*: We use the max-scand-time field to search for this element. If there is no such element in the tree, i.e. max-scand-time$[root] < t$, then return* NIL*. Otherwise we recursively descent the tree as follows: at each node n check whether max-scand-time$[left[n]] > t$. If so recursively descent to the left subtree. Otherwise, check whether time$[n] > t$ and scandalous?$[n]$ =* TRUE*. If so return n. Otherwise, recursively descent the right subtree.*