Midterm (version A) **15-451 Algorithms** **Fall 2007**

Name: ______________________ **Section:** __________

1	2	3	4	Total
43	20	21	16	100

One sheet of notes is allowed. Closed book.

---

1. **Multiple Choice: circle the correct answer (43 pts)**

   For (a)-(c), assume the base-case $T(x) = 1$ for $x \leq 3$.

   (a) The recurrence $T(n) = T(n/3) + T(n/2) + n$ solves to:

   $\Theta(1)$ $\Theta(\log n)$ $\Theta(n^{\log_3 2})$ $\Theta(n)$ $\Theta(n \log n)$

   (b) The recurrence $T(n) = T(2n/3) + 1$ solves to:

   $\Theta(1)$ $\Theta(\log n)$ $\Theta(n^{\log_3 2})$ $\Theta(n)$ $\Theta(n \log n)$

   (c) The recurrence $T(n) = 2T(n/3) + 1$ solves to:

   $\Theta(1)$ $\Theta(\log n)$ $\Theta(n^{\log_3 2})$ $\Theta(n)$ $\Theta(n \log n)$

   (d) Consider the following sorting algorithm: first insert all n elements into a B-tree with $t = 2$ (so this is a 2-3-4 tree). Then do an inorder traversal of the tree, to print everything out. This takes time:

   $\Theta(n)$ $\Theta(n \log n)$ $\Theta(n^2)$ $\Theta(n^2 \log n)$

   (e) Suppose we have a sorting algorithm that in addition to regular comparisons, is also allowed *super-comparisons*: a super-comparison takes in *three* elements and outputs those elements in order from smallest to largest. So, unlike a regular comparison that only has two possible outcomes, a super-comparison has 3! possible outcomes. Which of the following is a correct lower bound on the number of super-comparisons needed to sort an array of size n?

   $\log_2(n!)$ $\log_3(n!)$ $\log_6(n!)$ $n \log_2(n)$ n^2

(f) Al Gore-ithm (distant cousin of the former VP) gives you a data structure for a certain task with amortized cost $O(1)$ per operation. What does this amortized cost bound imply about a sequence of n operations? (Circle one).

(a) The total cost is $O(1)$ and each operation costs $O(1)$.
(b) The total cost is $O(n)$ and each operation costs $O(1)$.
(c) The total cost is $O(n)$ but a single operation might cost as much as $\Omega(n)$.
(d) The total cost is $O(n)$ but a single operation might cost as much as $\Omega(\log n)$.

(g) The *Hamming distance* between two n-bit vectors A and B is the number of locations i such that $A[i] \neq B[i]$. What is the expected Hamming distance between two *random* n-bit vectors (each location in each vector is determined by a fair coin flip)?

$$1 \qquad n/4 \qquad n/2 \qquad 3n/4 \qquad n$$

(h) Consider a random permutation of the numbers $1 \ldots n$. A number in this permutation is called a *Biggie* if it's greater than all the numbers to its left. (Note that the number that ends up in the first position is definitely a Biggie.)

- What's the probability that a number in position i is a Biggie? (The positions are numbered from left to right, starting from 1.)

$$1 \qquad 1/2 \qquad 1/\sqrt{i} \qquad 1/i \qquad 1/\sqrt{n} \qquad 1/n$$

- Building on your answer above, what's the expected number of Biggies in the whole array?

$$\Theta(1) \qquad \Theta(\log n) \qquad \Theta(\sqrt{n}) \qquad \Theta(n)$$

2. Truth or counterexample (20 pts). For each statement below, indicate whether it is true or false. If true, give a short proof. If false, give a counterexample.

(a) The order in which keys are inserted into a B-tree does not affect the final tree that is produced. That is, given a set of (distinct) keys, all insertion orders produce the same B-tree.

(b) Given a graph G, running Depth-First-Search, where you traverse edges in order of length, finds the MST. Specifically, the proposed algorithm is the following (starting from some arbitrary node v):

```
Min-DFS-tree(v):
  Mark v as visited.
  For each edge (v,w) in order from shortest to longest,
     If w is not marked,
       Put (v,w) into the tree
       Recursively run Min-DFS-tree(w)
```

(c) The optimal binary search tree for a sequence of lookup requests must have the most frequently-requested element at the root. (Recall from Homework 3 that the optimal binary search tree for a sequence of requests is the tree of least total cost.)

3. Dynamic Programming (21 pts). Given two sequences X and Y let $C(X, Y)$ denote the number of times that X appears as subsequence of Y. By *subsequence* we mean that the characters in X appear left-to-right in Y, but they do not have to be contiguous. For instance, the sequence AB appears 4 times as a subsequence of ADABCB. Let X_i denote the first i characters of the string X and let $X[i]$ denote the ith character (similarly for Y). Let m denote the length of X and let n denote the length of Y.

(a) Write a recurrence for $C(X_i, Y_j)$.

$$C(X_i, Y_j) = \begin{cases} \underline{\hspace{8cm}} & \text{if } X[i] \neq Y[j] \\ \underline{\hspace{8cm}} & \text{if } X[i] = Y[j] \end{cases}$$

Now set up the base cases so that your recurrence is correct.

$$C(X_0, Y_j) = \underline{\hspace{8cm}}$$

$$C(X_i, Y_0) = \underline{\hspace{8cm}} \text{ if } i > 0$$

(b) Let $C[i, j]$ be a 2-dimensional $m + 1$ by $n + 1$ matrix initialized to all -1s. Describe briefly, (or write pseudocode) how to convert your solution to part (a) into a dynamic programming algorithm to compute $C(X, Y)$. You may use either a bottom-up or top-down approach.

(c) What is the running time of your algorithm as a function of m and n (use O notation)

4. Hashing (16 pts) Let H be a set of k hash functions $\{h_1, \ldots, h_k\}$ mapping a universe U of size 2^n into the range $\{0, 1\}$. So, $M = 2$.

(a) Prove that if $k \leq n - 1$ then there must exist x and y in U ($x \neq y$) that collide under every hash function in H.

(b) Prove that if $k < 2(n - 1)$ then H cannot be a universal hash family. For instance, if U has size 8, then H needs to contain at least 4 functions. Hint: use part (a).