Practice Quiz 1

Problem 1. Algorithms and running times (5 parts) [5 points]

Match each algorithm below with the tightest asymptotic upper bound for its worst-case running time by inserting one of the letters A, B, . . ., E into the corresponding box. Some running times may be used multiple times or not at all. For sorting algorithms, $n$ is the number of input elements. For matrix algorithms, the input matrix has size $n \times n$.

You need not justify your answers. Because points will be deducted for wrong answers, do not guess unless you are reasonably sure.

- Insertion sort
- Binary Search
- BUILD-HEAP
- Strassen’s
- Randomized Quicksort

A: $O(\lg n)$
B: $O(n)$
C: $O(n \lg n)$
D: $O(n^3)$
E: $O(n^2)$
F: $O(n^{\lg 7})$
Problem 2. Recurrences (3 parts) [9 points]
Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

(a) \( T(n) = T(\sqrt{n}) + \Theta(\lg \lg n) \)

(b) \( T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046} \)

(c) \( T(n) = T(n/5) + T(4n/5) + \Theta(n) \)
Problem 3. Short Answers (4 parts) [16 points]

Give brief, but complete, answers to the following questions.

(a) Argue that you cannot have a Priority Queue in the comparison model with both the following properties.
   - EXTRACT-MIN runs in $\Theta(lg lg n)$ time.
   - BUILD-HEAP runs in $\Theta(n)$ time.

(b) A sequence of $n$ operations is performed on a data structure. The $i$th operation costs $i$ if $i$ is a power of two, and one otherwise. Determine the amortized cost per operation.

(c) What does it mean to sort in place, and what is one advantage of sorting in place? Which of the following algorithms sort in place?
   - INSERTION-SORT
   - MERGE-SORT
   - HEAPSORT
   - COUNTING-SORT
(d) If an algorithm has running time $T(m) \leq 2^m$ for all $m$ which are powers of 2, and $T(n)$ is monotonically increasing, then can we conclude that $T(n) = O(2^n)$ by using the sloppiness lemma?

(e) Consider the following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions, where the three hash functions map the universe $\{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$h_1(x)$</th>
<th>$h_2(x)$</th>
<th>$h_3(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$B$</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$C$</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$D$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Is this collection of hash functions universal?
Problem 4. True or False, and Justify (7 parts) [28 points]

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.

T F There exists a pivot selection algorithm such that quicksort runs in $O(n \lg n)$ time in the worst case.

T F Let $f$ and $g$ be asymptotically nonnegative functions. Then, at least one relationship of $f(n) = O(g(n))$ and $g(n) = O(f(n))$ must always hold.
T F Suppose we use a hash function $h$ to hash $n$ distinct keys into an array $T$ of length $m$. Assuming simple uniform hashing, the expected number of colliding pairs is $\Omega((\log n)/m)$.

T F Suppose that an array contains $n$ numbers, each of which is $-1$, 0, or 1. Then, the array can be sorted in $O(n)$ time in the worst case.
T F Suppose that a hash table of $m$ slots contains a single element with key $k$ and the rest of the slots are empty. Suppose further that we search $r$ times in the table for various other keys not equal to $k$. Assuming simple uniform hashing, the probability is $r/m$ that at least one of the $r$ searches probes the slot containing the single element stored in the table.

T F On all input arrays consisting of more than a 1000 elements, QUICKSORT performs at most as many comparisons as INSERTION-SORT.
T  F  Bucket sort can be used to sort an arbitrary list of real numbers in $O(n)$ expected time.
Problem 5. Sorting a partially-sorted array (3 parts) [10 points]

In this problem, more efficient algorithms will be given more credit. Partial credit will be given for correct but inefficient algorithms.

Let $A_0$ be a numerical array of length $n$, originally sorted into ascending order. Assume that $k$ entries of $A_0$ are overwritten with new values, producing an array $A$. Furthermore assume you have an array $B$ containing $n$ boolean values, where $B[i]$ is true if $A[i]$ is one of the $k$ values that was overwritten, and false otherwise.

(a) Give a fast algorithm to sort $A$ into ascending order, with time complexity better than $O(nk)$. [5 points]
(b) Give the time complexity of your algorithm in big-O notation, as a function of $n$ and $k$. [3 points]

(c) Give the space complexity of your algorithm in big-O notation, as a function of $n$ and $k$. (Do not include the space required for $A$ and $B$.) [2 points]
Problem 6. Tree Ancestors

Suppose you are given a complete binary tree of height $h$ with $n = 2^h$ leaves, where each node and each leaf of this tree has an associated “value” $v$ (an arbitrary real number).

If $x$ is a leaf, we denote by $A(x)$ the set of ancestors of $x$ (including $x$ as one of its own ancestors). That is, $A(x)$ consists of $x$, $x$’s parent, grandparent, etc. up to the root of the tree.

Similarly, if $x$ and $y$ are distinct leaves we denote by $A(x, y)$ the ancestors of either $x$ or $y$. That is,

$$A(x, y) = A(x) \cup A(y).$$

Define the function $f(x, y)$ to be the sum of the values of the nodes in $A(x, y)$.

Give an algorithm (pseudo-code not necessary) that efficiently finds two leaves $x_0$ and $y_0$ such that $f(x_0, y_0)$ is as large as possible. What is the running time of your algorithm?
SCRATCH PAPER — Please detach this page before handing in your exam.