Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- When the quiz begins, write your name on every page of this quiz booklet.
- The quiz contains five multi-part problems. You have 80 minutes to earn 80 points.
- This quiz booklet contains **5** pages, including this one.
- This quiz is closed book. You may use one handwritten $8\frac{1}{2}'' \times 11''$ crib sheet. 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	Initials
1	6	18		
2	3	15		
3	3	15		
4	4	20		
5	1	12		
Total		80		

Name: Solutions

Problem 1. Recurrences (6 parts) [18 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) $T(n) = 3T(n/5) + \lg^2 n$
Solution: By Case 1 of the Master Method, we have $T(n) = \Theta(n^{\log_5(3)})$.

(b) $T(n) = 2T(n/3) + n \lg n$
Solution: By Case 3 of the Master Method, we have $T(n) = \Theta(n \lg n)$.

(c) $T(n) = T(n/5) + \lg^2 n$
Solution: By Case 2 of the Master Method, we have $T(n) = \Theta(\lg^3 n)$.

(d) $T(n) = 8T(n/2) + n^3$
Solution: By Case 2 of the Master Method, we have $T(n) = \Theta(n^2 \log n)$.

(e) $T(n) = 7T(n/2) + n^3$
Solution: By Case 3 of the Master Method, we have $T(n) = \Theta(n^3)$.

(f) $T(n) = T(n-2) + \lg n$
Solution: $T(n) = \Theta(n \log n)$. This is $\sum_{i=1}^{n/2} \lg 2i \geq \sum_{i=1}^{n/2} \lg i \geq (n/4)(\lg n/4) = \Omega(n \lg n)$. For the upper bound, note that $T(n) \leq S(n)$, where $S(n) = S(n-1)+\lg n$, which is clearly $O(n \lg n)$.

Problem 2. True or False, and Justify (3 parts) [15 points]

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

(a) **T** **F** If $f(n)$ does not belong to the set $o(g(n))$, then $f(n) = \Omega(g(n))$.

Solution: False. For example $f(n) = n^{\sin(n)}$ and $g(n) = n^{\cos(n)}$.

(b) T F An adversary can construct an input of size n to force RANDOMIZED-MEDIAN to run in $\Omega(n^2)$ time.

Solution: False. The *expected* running time of RANDOMIZED MEDIAN is $\Theta(n)$. This applies to *any* input.

(c) T F A set of n integers in the range $\{1, 2, \ldots, n\}$ can be sorted by RADIX-SORT in $O(n)$ time by running COUNTING-SORT on each bit of the binary representation.

Solution: False. This results in $\Theta(\log n)$ iterations of counting sort, and thus an overall running time of $\Theta(n \log n)$.

Problem 3. Short Answer (3 parts) [15 points]

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

(a) Consider any priority queue (supporting INSERT and EXTRACT-MAX operations) in the comparison model. Explain why there must exist a sequence of n operations such that at least one operation in the sequence requires $\Omega(\lg n)$ time to execute.

Solution: Take an array A of $n/2$ elements. Insert the $n/2$ elements into the priority queue, and then extract-max $n/2$ times. This sorts the array à la Heapsort. If none of the operations in this sequence require $\Omega(\lg n)$ time, then these n operations take $o(nlogn)$ time, which is impossible in the comparison model, since we've shown an $\Omega(n/2 \log(n/2)) = \Omega(n \log n)$ lower bound for any sorting algorithm.

(b) Suppose that an array A has the property that only adjacent elements might be out of order—i.e., if $i < j$ and $A[i] > A[j]$, then $j = i + 1$. Which of INSERTION-SORT or MERGE-SORT is a better algorithm to sort the elements of A? Justify your choice.

Solution: INSERTION-SORT is a better choice: since its running time depends on the number of inversions in the array. We may swap each element with the element immediately to its left, but that's all that we do in INSERTION-SORT. So INSERTION-SORT is $O(n)$ in this case, while MERGE-SORT remains $\Theta(n \log n)$.

(c) Suppose that a hash table with $3n$ slots resolves collisions by chaining, and suppose that $n/4$ keys are inserted into the table. For $i = 1, 2, \ldots, n/4$ and $j = 1, 2, \ldots, 3n$, define the indicator random variable X_{ij} as

$$X_{ij} = \begin{cases} 1 & \text{if element } i \text{ hashes to slot } j \text{ ,} \\ 0 & \text{otherwise .} \end{cases}$$

Under the assumption of simple uniform hashing (each key is equally likely to be hashed into each slot), give a formula in terms of the X_{ij} for the expected number of keys that fall into slot 1, and solve your equation.

Solution: $E[X_{ij}] = Pr[X_{ij} = 1] = 1/(3n)$. Then the expected number of elements in slot 1 is $E[\sum_{i=1}^{n/4} X_{i1}] = \sum_{i=1}^{n/4} E[X_{i1}] = n/(4(3n)) = 1/12$ by linearity of expectation.

Problem 4. Finding the smallest elements of an array (4 parts) [20 points]

In this question, we will explore several algorithms for finding the k smallest elements of an array of n integers, in sorted order. For example, given the array `[5 2 1 9 6 7 3 4 8]` and $k = 3$, such an algorithm should return `[1 2 3]`.

(a) Algorithm A sorts the numbers using merge sort and outputs the first k elements in the sorted order. Analyze the worst-case running time of Algorithm A in terms of n and k.

Solution: MERGE-SORT requires $O(n \log n)$ time to sort, and the extraction can be done in $O(1)$ time each. Then the overall running time is $\Theta(n \log n + k)$.

(b) Algorithm B builds a min-heap from the numbers and calls EXTRACT-MIN k times. Analyze the worst-case running time of Algorithm B in terms of n and k.

Solution: BUILD-HEAP can be implemented in $O(n)$ time, and EXTRACT-MIN takes $O(\log n)$ time. Thus overall the running time is $\Theta(n + k \log n)$.

(c) Algorithm C uses SELECT to find the kth-largest element in the array, partitions around that number, and then insertion-sorts the k smallest numbers. Analyze the worst-case running time of Algorithm C in terms of n and k.

Solution: We can SELECT and PARTITION in $\Theta(n)$ time. Then insertion-sorting k numbers takes $\Theta(k^2)$ time. Overall the running time is $\Theta(n + k^2)$.

(d) Briefly describe an algorithm that is asymptotically at least as good as Algorithms A, B, and C, and give its running time. You need not argue correctness.

Solution: Do exactly the same thing as in Algorithm C, but use, say, mergesort instead of insertion sort. This gives a $\Theta(n + k \log k)$ running time, which asymptotically beats all three of A, B, C.

Problem 5. Mean 6.042 instructors (1 parts) [12 points]

There are two types of professors who have taught 6.042: ***nice*** professors and ***mean*** professors. The nice professors assign A's to all of their students, and the mean professors assign A's to **exactly** 75% of their students and B's to the remaining 25% of the students. For example, for $n = 8$, the arrays [A A A B A B A A] and [A B B A A A A A] represent grades assigned by mean professors, and the array [A A A A A A A A] represents grades assigned by a nice professor. Given an array $G[1..n]$ of grades from 6.042, we wish to decide whether the professor who assigned the grades was nice or mean.

Give an efficient **randomized** algorithm to decide whether a given array G represents grades assigned by a mean or nice professor. Your algorithm should be correct with probability at least 51%.

Solution: Here's the algorithm:

1.Repeat the following t times:

(a)choose a random index $i \in \{1, \ldots, n\}$.

(b)if $G[i] = B$, then return "mean"

2.Return "nice".

First, note that with probability one, a nice professor will be noted as such, since there are no B's in G in this case.
The probability that a mean professor makes it through an iteration of the loop is $3/4$, since $1/4$ of the entries of array are B's. The probability that the mean professor makes it through t iterations is $(3/4)^t$. If we take t to be 3, then $(3/4)^3 < 0.49$. Thus the probability of giving a correct answer is at least 51%.
The running time is obviously $O(1)$, since we do a constant number of iterations of constant-time loop.

Comment: The above arguments show that

- If a professor is nice, the probability of an incorrect answer is 0
- If a professor is mean, the probability of an incorrect answer is $\leq 49\%$

However, in several cases, people were making "reverse" statements, e.g.,:

"If we get an A, then the probability that the professor is mean is ..."

Although intuitive, a closer inspection reveals that such a statement does not have any real meaning. This is because there is *no* probability distribution determining if the professor is mean or not. In case of randomized algorithms, the input is *not* random. Instead, for *any* input, the algorithm should report the correct answer with at least certain probability.