Sorting (2 points)

1. [2 points] We can express insertion sort as a recursive procedure as follows. In order to sort $A[1 \ldots n]$, we recursively sort $A[1 \ldots n-1]$ and then insert $A[n]$ into the sorted array $A[1 \ldots n-1]$. Write a recurrence for the running time of this recursive version of insertion sort.

Solution: since it takes $\Theta(n)$ time in the worst case to insert $A[n]$ into the sorted array $A[1 \ldots n-1]$, we get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n-1) + \Theta(n) & \text{if } n > 1. \end{cases}$$

The solution to this recurrence is $T(n) = \Theta\left(n^2\right)$

Merge Sort (6 points)

2. Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta\left(n^2\right)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

   (a) [1 point] Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

   Solution: The time for insertion sort to sort a single list of length k is $\Theta\left(k^2\right)$, so, n/k of them will take time $\Theta\left(\frac{n}{k}k^2\right) = \Theta(nk)$.

   (b) [2 points] Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.

   Solution: Just extending the 2 -list merge to merge all the lists at once would take $\Theta(n \cdot (n/k)) = \Theta\left(n^2/k\right)$ time (n from copying each element once into the result list, n/k from examining n/k lists at each step to select next item for result list).
   To achieve $\Theta(n \lg(n/k))$-time merging, we merge the lists pairwise, then merge the resulting lists pairwise, and so on, until there's just one list. The pairwise merging requires $\Theta(n)$ work at each level, since we are still working on n elements, even if they are partitioned among sublists. The number of levels, starting with n/k lists (with k elements each) and finishing with 1 list (with n elements), is $\lceil \lg(n/k) \rceil$. Therefore, the total running time for the merging is $\Theta(n \lg(n/k))$

   (c) [2 points] Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

Solution: The modified algorithm has the same asymptotic running time as standard merge sort when $\Theta(nk + n\lg(n/k)) = \Theta(n\lg n)$. The largest asymptotic value of k as a function of n that satisfies this condition is $k = \Theta(\lg n)$.

To see why, first observe that k cannot be more than $\Theta(\lg n)$ (i.e., it can't have a higher-order term than $\lg n$), for otherwise the left-hand expression wouldn't be $\Theta(n\lg n)$ (because it would have a higher-order term $n\lg n$). So all we need to do is verify that $k = \Theta(\lg n)$ works, which we can do by plugging $k = \lg n$ into $\Theta(nk + n\lg(n/k)) = \Theta(nk + n\lg n - n\lg k)$ to get

$$\Theta(n\lg n + n\lg n - n\lg\lg n) = \Theta(2n\lg n - n\lg\lg n)$$

which, by taking just the high-order term and ignoring the constant coefficient, equals $\Theta(n\lg n)$.

(d) [1 point] How should we choose k in practice?

Solution: In practice, k should be the largest list length on which insertion sort is faster than merge sort.

A more precise (but not really needed in this course) answer: If we optimize the previous expression using our calculus 1 skills to get k (by getting the derivative and setting it to 0 to find the minimum), we have that $c_1 n - \frac{nc_2}{k} = 0$ where c_1 and c_2 are the coefficients of nk and $n\lg(n/k)$ hidden by the asymptotics notation. In particular, a constant choice of k is optimal.

Quicksort (5 points)

3. [1 point] What is the running time of quicksort when all elements of array A have the same value?

Solution: The running time of quicksort on an array in which every element has the same value is n^2. This is because the partition will always occur at the last position of the array (Exercise 7.1-2) so the algorithm exhibits worst-case behavior.

4. [2 points] When RANDOMIZED-QUICKSORT runs, how many calls are made to the random number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.

Solution: In the worst case, RANDOM returns the index of the largest element each time it's called, so $\Theta(n)$ calls are made. In the best case, RANDOM returns the index of the element in the middle of the array and the array has distinct elements, so $\Theta(\lg n)$ calls are made.

5. [2 points] Suppose that the splits at every level of quicksort are in the proportion $1 - \alpha$ to α where $0 < \alpha \leq 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion

tree is approximately $-\lg n / \lg \alpha$ and the maximum depth is approximately $-\lg n / \lg(1-\alpha)$. (Don't worry about integer round-off.)

Solution: The minimum depth corresponds to repeatedly taking the smaller subproblem, that is, the branch whose size is proportional to α . Then, this will fall to 1 in k steps where $1 \approx (\alpha)^k n$. So, $k \approx \log_\alpha(1/n) = -\frac{\lg(n)}{\lg(\alpha)}$. The longest depth corresponds to always taking the larger subproblem. we then have and identical expression, replacing α with $1-\alpha$.