CSE 5311 Homework Assignment 1 (Fall 2019)

Due date: 9/4 (Wednesday) (type, print, and hand-in in class)

(1) [60 points]
Implement the insertion sort and merge sort algorithms with any programming language you choose and run them with input number lists. Generate the list elements with a random function and increase the list size incrementally until you find the execution time of your merge sort program is consistently shorter. Draw the two curves in a figure (execution time vs input list size) about the two programs. Using the example to discuss why the asymptotic analysis is meaningful. Attached program codes in your submission.

Answer:
[Your code and figure showing execution times with appropriately chosen list sizes.]

The figure is supposed to show that the insertion sort runs faster than the merge sort for small input sizes. When the input size n becomes large enough, the merge sort performs always better than insertion sort.

Asymptotic analysis allows us to gauge and compare performance of an algorithm when the problem size grows and when the size is sufficiently large. It provides a guideline on selection of an algorithm out of a set of candidates for a problem to solve the problem with sufficiently large problem size.

(2) **[40 points]** Problem 2-1 on Page 39 of the CLRS textbook.
2-1 Insertion sort on small arrays in merge sort
Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ 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. Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.
b. Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.
c. 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?
d. How should we choose k in practice?

Answer:

a. Insertion sort takes $\Theta(k^2)$ time per k-element list in the worst case. Therefore, sorting n/k lists of k elements each takes $\left(\frac{n}{k}\right) * \Theta(k^2) = \Theta(nk)$ worst-case time.

b. Just extending the 2-list merge to merge all the lists at once would take $\Theta(n \cdot (n/k)) = \Theta(n^2k)$ 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. 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 than $n\lg n$). 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. In practice, k should be the largest list size on which insertion sort is faster than merge sort.