Exercises for Tutorial 2

Carl Leonardsson Jari Stenman

2013-09-13

Exercises

1. Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n-1$ elements of A. Write pseudo code for this algorithm, which is known as **selection sort**. Why does it need to run for only the first $n-1$ elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ-notation. [CLRS 2.2-2 (2nd ed)]

2. Use the Master Theorem to find tight asymptotic bounds for the following recurrences. For each recurrence state the bound, and which case of the Theorem that applies. **For some recurrences the Master Theorem does not apply. In those cases, just state that MT does not apply.**

   (a) $T(n) = 2T(\frac{n}{2}) + \Theta(1)$
   (b) $T(n) = T(n-1) + n^2$
   (c) $T(n) = T(\frac{n}{2}) + 3n$
   (d) $T(n) = 3T(\frac{n}{3}) + \Theta(n \log n)$
   (e) $T(n) = 2T(n\frac{2}{3}) + \Theta(n)$
   (f) $T(n) = 4T(\frac{n}{2}) + 2n^2 - n$

3. How does quicksort perform on an array where all elements are the same value? What happens? What is the asymptotic bound on the runtime in this case? How about the case when there are only two distinct values, and each element in the array takes on one of those values? *No proof required, only reasoning.*

4. Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Argue that the procedure INSERTION-SORT would tend to beat the procedure QUICKSORT on this problem. [CLRS 7.2-4 (2nd ed)]

5. The running time of quicksort can be improved in practice by taking advantage of the fast running time of insertion sort when its input is “nearly” sorted. When quicksort is called on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process. Argue that this sorting algorithm runs in $O(nk + n \log(n/k))$ expected time. and in practice? [CLRS 7.4-5 (2nd ed)]