Homework 1:
Searching, Sorting and Order Statistics

udayan@codito.com Saturday 04 September 2004

1. In binary search, if instead of asking "Which half of the remaining class has my keychain?" I had asked "Do you have my keychain? Which half of the remaining class has my keychain?" what would be the (a) best case, (b) worst case and (c) average time for the algorithm?

2. Suppose the elements we are searching on are arranged in a 2D square grid, and we are searching using the question "Do you have my keychain? Who among your (4) neighbors is closer to the keychain than you are?" What is the best point to start the search? Why?

3. Suppose the elements we are searching on are arranged in a d-dimensional hyper-square, and we are searching using the question "Do you have my keychain? Who among your ($2d$) neighbors is closer to the keychain than you are?" Suppose we start from a corner. What is the worst case search time?

Assume that an element takes as much time to answer as he/she has neighbors. Now what is the worst case time? (You may disregard edge effects, i.e. lesser neighbors on the edges.) Suppose you know n, the number of elements, what is the optimal dimensionality d to use?

4. Suppose each element is twice as likely to say "Yes" as the next, what is the average time taken by sequential search?

5. Suppose a single "bubble pass" of bubble sort takes pn worst case time for passing over n elements, whereas a "merge pass" of merge sort takes qn worst case time to merge n elements. Find the range of n where you would prefer bubble sort over merge sort, assuming the worst case. Suppose we implemented merge sort as a recursive algorithm, which switches to bubble sort "where beneficial", what would be the running time of this sorting algorithm? (Hint: it may not be beneficial to switch to bubble sort the moment bubble sort is beneficial. Explain this paradox.)

6. Suppose our sorting algorithm gets repeated sorting requests, each for n elements. Suppose the inversion characteristics of the data change slowly – if there are u inversions in a given input set, the next input set will have $u \pm 10\%u$ inversions. Suppose our basic algorithm is merge sort (timed at $qn\log_2 n$) which switches to insertion sort (timed at $pu + rn$) whenever low number of inversions are expected. (a) What should the present u be for us to switch over to insertion sort for the *next* input? (b) Modify the merge sort and insertion sort algorithms to also measure the number of inversions. (c) We implement the above scheme and find that q, p and r all increased because of our modifications to q_1, p_1 and r_1. Should we still implement this scheme, or use just plain merge sort?

7. Merge sort is an external algorithm, in the simplest implementation taking $n \log_2 n$ extra memory. Can you make merge sort take only n memory? Only $n/2$ extra memory? Remember to optimize your algorithms after devising them.

8. Suppose we have 500 numbers, all integers between 1 and 100. Devise a stable (!) yet internal quick sort to sort them. (Assume the machine uses 16 bit integers.) (Hint: think about how *any* sorting algorithm can be converted to a stable sort.)

9. A decision tree is a binary tree, each of whose nodes has either two children or none. Prove that in a decision tree, the deepest leaf has a sibling leaf. Prove that among all decision trees with p leaves, the decision tree having least average leaf depth is essentially complete[1]. (Hint: proof by contradiction.) Prove that for any decision tree with p leaves, the average leaf depth can be no better than $\log_2 p$. Prove that assuming all permutations equally likely, no comparison sorting algorithm can work faster than $\theta(n \log_2 n)$ on the average.

10. A particular approximate median algorithm on n numbers works by finding the exact medians of $\sqrt{n}$ sets of size $\sqrt{n}$ each, and then finding the exact median of these $\sqrt{n}$ medians. What percentile cut is guaranteed by this approximate median algorithm? Suppose the exact median algorithm uses this approximate median algorithm recursively to find the pivot, analyze the running time of the exact median algorithm.

11. Calculate (non-asymptotic) bounds on the average number of comparisons performed by quick-order-statistic and quick-order-statistic with median-of-5-medians pivot. I devise the following scheme for quick-order-statistic. I first do a random pivot, and perform separation. If the size of the set I am descending into is larger than $f(n)$ then I perform median-of-5-medians pivot. Design $f(n)$ such that my worst case is $\theta(n \log n)$ and the average number of comparisons is at most three times that of quick-order-statistic.

12. Suppose I have a pivoting methodology which will guarantee at least $\sqrt{n}$ elements on either side of the pivot. What is the worst case running time of quick-sort and quick-order-statistic using this pivoting methodology? (Assume the pivoting algorithm is extremely fast.) Devise a pivoting algorithm that will guarantee at least $\sqrt{n}$ elements on either side of the pivot. Analyze quick-order-statistic's running time using this pivoting methodology.

[1] An essentially complete binary tree is one which has leaves only on two successive levels.