Algorithms – Examples Sheet

Alan Mycroft
`http://www.cl.cam.ac.uk/users/am`

While these problems focus on a small area of algorithms, they touch on many issues about algorithms and timing.

There are probably far too many questions here – start with the ones which look interesting to you (the questions vary hugely in difficulty and are designed to make you think).

1. *Finding the minimum and maximum of a list.*
   It's easy to do this in $(n-1)+(n-2)$ comparisons (first find the maximum and then the minimum of the remainder). But, if n is even, can you see how to do it in $n/2+(n/2-1)+(n/2-1)$ comparisons? What if n is odd?

2. *Finding the minimum and 2nd-minimum of a list.*
   Again, we could do this in $(n-1)+(n-2)$ comparisons based on "finding the smallest" from lectures. However, suppose we use another algorithm for finding the smallest: arrange the numbers like a knock-out tournament (FA cup or Wimbledon according to taste); it makes things easier to think about if n is a power of two. Now how many comparisons are needed to find the smallest? After the knockout has found the smallest, how many more comparisons are needed to find the next smallest? [Hint: far less than $n-2$.]

3. *Better ways for searching.*
   The lectures gave an algorithm for searching for a *key* in a list of n elements which took n comparisons in the worst-case. If the list was already sorted (think of a telephone directory) then obviously humans don't do n (or even $n/2$ on average) comparisons of names. How many comparisons need it now take?

4. *Insertion sort.*
   Here's another way of sorting. Keep a sorted list of the 'answer so far', initialised to 'empty'. Then take each element from the input list in turn and insert it (using the 'fast search' from the previous question). Estimate the number of comparisons required. [Additional (subtle) question: is it fair to count only comparisons here?]

5. *How bad is "find all permutations and pick (the) sorted one"?*
   Lectures mentioned this as a (slow) algorithm. Estimate the number of comparisons needed to sort using this method, both in the best case and the worst case. Ignore any administration like forming all the permutations, and just count comparisons. [Additional question: is this fair?] What about the word 'the' in the question title?

6. *Is there a slowest method of sorting?*
   This is rather open-ended. Obviously we can sort things twice (or even a million times), but doing an $O(n^2)$ algorithm twice is still $O(n^2)$ (remember we ignore constant speed factors). Also calculating things and ignoring the result is kind-of cheating. Can we find a sorting algorithm which goes slower than that of the previous question?

7. *Design input to quicksort which makes it take $n(n-1)/2$ comparisons (therefore showing it's not quick at all).*
   [Obviously only if you've seen quicksort before!]

8. (For next lecture) *Try to factorise 1591 or 25957 into a product of primes by hand.*