Homework 1
Due Oct 7, 2003

Amit Chakrabarti
Department of Computer Science
Dartmouth College

Each of these problems requires you to prove something. Please think carefully about how you are going to organise your proof *before* you begin writing. Each problem has a fairly short solution; if you find yourself writing a page or more of proof, you have probably not found the best solution. So retry the problem.

1. Consider the problem of selecting both the minimum and the maximum of n given numbers, using comparisons. The naive solution would be to first find the minimum and then the maximum, thereby using nearly $2n$ comparisons. But, as we remarked in class, there is an algorithm which solves this problem making only $3\lceil n/2 \rceil$ comparisons.

   Your task is to prove an almost matching lower bound. Prove that any comparison based algorithm for this problem *must* use at least $\lceil 3n/2 \rceil - 2$ comparisons.

2. We would like to sort an array of numbers using only in-place exchanges of the form `xch(i,j)` which swaps the array elements at indices i and j. Furthermore, we would like to avoid exchanging elements that are "far away": the two indices involved in an exchange must differ by no more than some constant C. For instance, "bubble sort" is an exchange-based sorting algorithm and it only swaps adjacent pairs, so it satisfies our requirement with $C = 1$.

   Prove that under these conditions sorting requires $\Omega(n^2)$ time in the worst case. Be careful! Note that we did not insist that the algorithm uses comparisons alone.

3. Consider the problem of finding the second largest of n given elements (numbers, if you prefer) using comparisons. In class, we used a *leaf counting argument* to prove a lower bound of $n - 2 + \log n$ comparisons. This exercise will walk you through an alternative proof of this lower bound using an *adversarial argument*.

   The algorithm asks queries of the form "is $x_i < x_j$?" and the adversary answers them using an *adversarial strategy* which satisfies two properties:

   (a) There is always at least one input with which the adversary's answers are consistent.

   (b) If the algorithm has asked less than $n - 2 + \log n$ questions so far, then there are at least two consistent inputs with different answers (for the second largest element).

   Clearly the existence of such a strategy proves the lower bound.

   The adversary maintains n *tokens*, initially allocated one per element. When he* is asked a comparison query between two elements, he declares the element that has more tokens the winner, and then he takes away all of the loser's tokens and gives them to the winner.

   3.1. Spell out this adversarial strategy in detail (make sure you handle all cases) and argue that it satisfies property (a) above.

   3.2. Recall that we argued in class that by the time the algorithm knows the second largest element, it must also know the largest. Prove that when the algorithm finds out the largest element, the adversary must have allocated all n tokens to that element.

   3.3. Prove that the largest element must therefore win at least $\log n$ comparisons.

   3.4. Finish the lower bound proof by showing that property (b) above is satisfied. Don't try to actually produce two different consistent inputs; just argue that having performed less than $n - 2 + \log n$ comparisons the algorithm cannot know the answer for sure.

4. For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, prove the following relation between its certificate complexity, its sensitivity and its block sensitivity: $\mathrm{C}(f) \leq \mathrm{s}(f)\mathrm{bs}(f)$. Hint: fix an input to the function and consider a set of disjoint sensitive blocks each of which is minimal. How large can each block be? How many blocks can there be, at most? Now produce a short certificate.

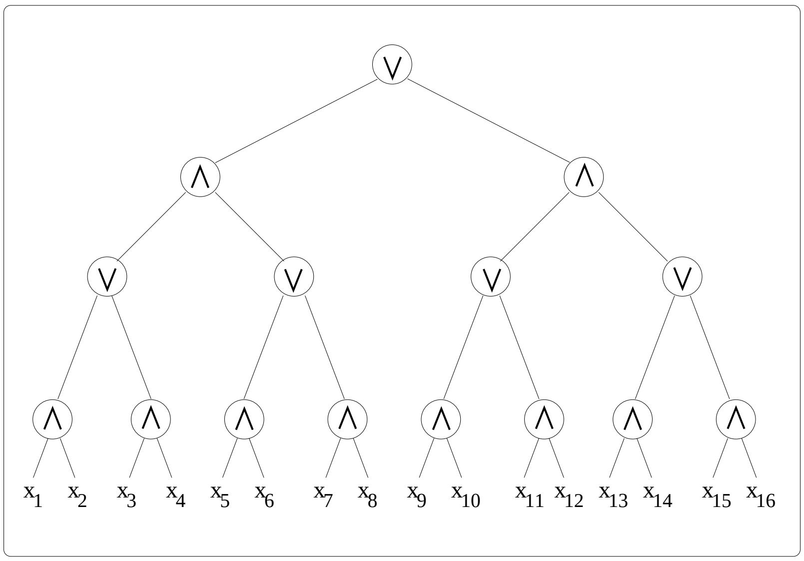
*It is considered ungallant to use "she" for a character as mean as the adversary!

5. In this problem we shall prove two lower bounds on the (asymptotic) gap between some of the complexity measures for Boolean functions that we have studied.

   5.1. Construct a family of functions $f_n : \{0,1\}^n \to \{0,1\}$ such that, for infinitely many n, $\mathrm{D}(f_n) = \Theta(n)$ whereas $\mathrm{C}(f_n) = \Theta(\sqrt{n})$. This shows that the gap between $\mathrm{C}(f)$ and $\mathrm{D}(f)$ can be quadratic. We have already shown in class that $\mathrm{D}(f) \leq \mathrm{C}(f)^2$ for any f, so this is the largest possible gap.

   5.2. Construct a family of functions $g_n : \{0,1\}^n \to \{0,1\}$ such that, for infinitely many n, $\mathrm{bs}(g_n) = \Theta(\sqrt{n})$ whereas $\deg(g_n) = \Theta(n)$. Again, this shows a quadratic gap. From what we have seen in class, and from Problem 4, we have $\deg(f) \leq \mathrm{D}(f) \leq \mathrm{C}^{(0)}(f)\mathrm{bs}(f) \leq \mathrm{s}(f)\mathrm{bs}(f)^2 \leq \mathrm{bs}(f)^3$; therefore the gap can be at most cubic.

A hint for the first sub-problem: Consider only those n's which are powers of 2. Then consider an appropriate generalisation of the function in the figure below.



A hint for the second sub-problem: Consider only those n's which are perfect squares and divide the n inputs into $\sqrt{n}$ equal-sized blocks.