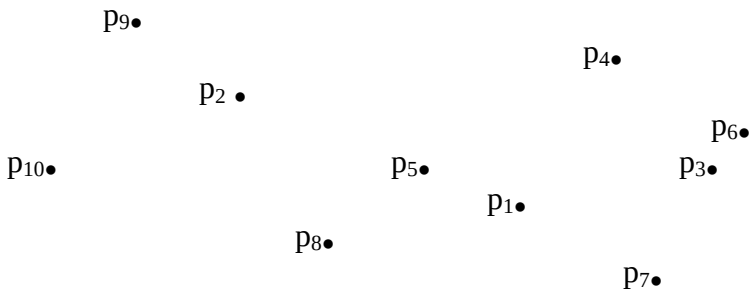
CS 466/666
Assignment 2
Due: Noon Monday October 18, 2010
Justify all answers

1) [4 marks] Prove $(\lg\lg n)^{\lg\lg n} = (\lg n)^{\lg\lg\lg n}$ (Now sing the equation 5 times as fast as you can: no need to hand in the vocals.)
2) [8 marks] Let $P = (p_0 , p_1 , \ldots , p_{n-1}, p_n = p_0)$ be a non-convex polygon which is simple (its edges $p_i p_{i+1}$ do not intersect). Describe a linear-time algorithm that computes the convex hull of P.
3) [14 marks] We are given a set $P = (p_0 , p_1 , \ldots , p_{n-1}, p_n)$ of n points in the plane, where $p_i = (x_i, y_i)$, x_i and $y_i > 0$. We say that a point p_i is above a point p_j if $y_i > y_j$. We define $\text{lft}(p_i)$ as the rightmost point in P that is above and to the left of p_i (if such a point exists). Similarly, we define $\text{rt}(p_i)$ as the leftmost point in P that is above and to the right of p_i (if such a point exists). In the example of Figure 1 we have $\text{lft}(p_1) = p_5$, $\text{rt}(p_1) = p_4$, and $\text{lft}(p_4) = p_9$. Also $\text{rt}(p_4)$ and $\text{lft}(p_9)$ are not defined. We want to find $\text{lft}(p_i)$ and $\text{rt}(p_i)$ for all points p_i.
   a) Upper Bound: Give an algorithm(in a reasonable pseudocode) that solves this problem in O(n log n) time. Hint: Use an incremental approach similar to Graham Scan. Sort the points in x-coordinate and consider them one by one from left to right.
   b) Lower Bound: Prove that any comparison-based algorithm requires Θ(n log n) time to solve this problem. Hint: You may assume a lower bound for sorting of Θ(n log n).



4) [10 marks] Consider the van Emde Boas structure as discussed in class. We included a modification to that the space requirement was O(u) bits; enhance this approach to give a structure requiring u +o(u) bits while still supporting the operations in O(lg lg u) time. Justify your claim.
5) [18 marks] This question deals with binary search on a sorted array of n values and the number of cache line misses. Memory is divided into cache lines of size c consecutive words; when a word is accessed it and its line may or may not be in the cache; and if not the dominant cost is getting the line into cache. On top of this, there are several levels of cache with lines of different sizes (including pages coming from disk), so we don't necessarily know the value of c that is causing our main problem. Assume, though, that n is substantially larger than c. A[i] denotes the i^{th} smallest value. Note that the comparisons performed do not depend on the layout. The issue is that a "good layout" will allow us to perform "several" comparisons for each cache miss. A "bad layout" will permit only one (most of the time).

a) If our array is stored in the usual sorted order, how many cache misses will be made?
b) If our array is stored with the median (i.e. A[n/2]) in position 1; elements A[n/4] and A[3n/4] in positions 2 and 3, followed by the other 4 elements if ranks (2i-1)n/8, the next 8 locations are occupied by elements of ranks (2i-1)n/16, etc.; how many cache misses occur in a binary search. How many cache misses do we have on a binary search?
c) If the array is stored according to the "van Emde Boas layout" in the following sense. $n^{½}$ equally spaced (by rank) elements are put in the first $n^{½}$ positions; followed by the smallest $n^{½}$, the next $n^{½}$ etc. This layout is applied recursively so the layout has the following recursive form. Here the values (A[i]) correspond to the implicit balanced binary tree we would have in any binary search, but the number in front of them gives the memory locations (starting at1)

1:A[n/2]

2:A[n/4] 3:A[3n/4]

4:A[n/8] 7:A[3n/8] 10:A[5n/8] 13:A[7n/8]

5:A[n/16] 6:A[3n/16] …etc

16:A[n/32] 31:A[3n/32]

i.e. the median element is in the first position, the element of rank 7n/8 is in the 7^{th} position, that of rank n/16 and its subtree of size 15 (A[n/16], A[n/32], A[3n/32], A[n/64], A[n/128], A[3n/128], A[3n/64] etc) come next etc. In this example if the cache line size were 2, we would have 3 misses to find A[3n/16], and 4 for A[3n/32]. How many cache misses occur in a binary search (with this layout) in general? (Here an answer up to Θ notation is adequate.) You may find it useful to first determine the best layout (and number of cache misses) for a given value of c, but the question is to determine the number of cache misses using a fixed layout … the van Emde Boas layout.