CS 667 : Homework 1(Due: Feb 7, 2012)

This Homework is Problem 1-6 and worth 250 points. You can replace some of these points with Problem 7 or 8, but you can only submit 250 points worth of problems.

Problem 1. (50 points)
(a) Professor I.M. Nuts guesses a positive integer n in the range $1 \ldots N$, i.e. $1 \leq n \leq N$. What is the minimal number of questions t, and what type of questions can you ask him to get this number n? Justify your answer and show it is optimal. Express t as a function of N.
(b) Dr I.M. Nuts guesses again a positive integer n but in this case you have no information about the range of n i.e. you know nothing about N. How small can t be now? Justify your answer. Can you still achieve a bound for t of the order of part (a)? Explain.
Hint for (b): Guess the N by guessing the number of bits of n...

Problem 2. (25 points)
Can you sort 5 keys with 7 comparisons in the worst-case? Explain.
Hint: Binary search or merging can help.

Problem 3. (50 points)
This problem assumes a WORD model of computation. The Fibonacci sequence is given by the following recurrence $F_{n+1} = F_n + F_{n-1}$ for $n \geq 1$ and $F_0 = F_1 = 1$.
(a) Show how to compute F_n in $O(n)$ time, and $O(n)$ space.
(b) Shwo how to compute F_n in $O(n)$ time and $\Theta(1)$ space.
(c) Given a $k \times k$ matrix A show how you can find A^n in $O(k^3 \lg n)$ time.
(d) Can you improve the obvious time bound in (a)/(b)? In particular prove that F_n can be computed in $O(\lg n)$ time.
Hint: You may need to use the result of part (b), i.e. formulate the F_n as a matrix problem. The discussion of problem 31-3 (CLRSE3e) in the Problem section of the Chapter on Number-Theoretic Algorithms might offer you some insight on this.

Problem 4. (25 points)
You are given an array of n keys $A[1 \ldots n]$. We are interested in finding the $m \ll n$ keys closest to the median of the n keys. (Be reminded that the median is the $\lfloor (n+1)/2 \rfloor$ order statistic of A.) Give a linear-time worst-case algorithm that solves this problem.

Problem 5. (50 points)
(a) Professor I.M. Nuts is about to decide the letter grades for his n students. Numeric grades are stored in array $A[1 \ldots n]$. Professor Nuts wants to split A into five disjoint subsets A_1, A_2, A_3, A_4, A_5 of the same size such that all grades in A_1 are less than or equal to those in A_2; the grades in A_2 are less than or equal to those in A_3 and so on. (The grades in any subset need not be ordered.) Then Professor Nuts would assign a letter grade A to all students in A_5, a B+ to those in A_4 and so on. Give an algorithm that can effect this split into A_1, A_2, A_3, A_4, A_5 in linear time. (You may assume n is a multiple of 5.)
(b) An array of n distinct keys $A[1 \ldots n]$ is m-sub-sorted if it consists of m subarrays each of size n/m such that the keys of each subarray are smaller than keys of later (to the right) subarrays and greater than the keys of earlier (to the left) subarrays. For example the following array is 4-sub-sorted and the array in (a) was to become 5-sub-sorted.

```
4  2  1  3  5  6  7  8  12 10 11  9  16  15 14 13
```

Give an algorithm that for a an array A that is m-sub-sorted array, it completely sorts it in worst-case time $O(n \lg (n/m))$.
(c) Give an algorithm that m-sub-sorts an arbitrary array in worst-case time $O(n \lg m)$.

Turn page

Problem 6. (50 POINTS)
For this problem the WORD model of computation allows only the standard arithmetic operations `+,-,*,/` in UNIT time but not square roots, cubic roots etc. Other non-arithmetic operations are allowed as shown below (eg. iterative and branching statements).

```
  PerfectPower(n)    // Pseudocode that looks like C/C++ or Java
1.   XX=n;
2.   YY=n;
3.   for(x=2 ; x <= XX ; x++)
4.      for(y=2 ; y <= YY ; y++)
5.        if ( RaiseToPower(x,y) == n )
6.          print x,y;
7.          return;
8.        endif
9.      endfor
10.  endfor
11.  printline 'NO';
```

Function `PerfectPower` is an attempt to solve the following perfect integer problem. Given an integer n determine whether there exist integers $x \geq 2, y \geq 2$ such that $x^y = n$. If no such integers exist a `NO` is being printed, otherwise a single pair of x, y is printed and the function terminates.

There are two things missing in `PerfectPower`: (i) an efficient implementation of `RaiseToPower(x,y)` that returns x^y, i.e. x raised to the y-th power, and (ii) a worst-case running time analysis of `PerfectPower`. Even with these realizations, `PerfectPower` won't be optimal.

Let's start with some easy questions related to this non-optimal implementation above.

(a) What is a tight upper bound for `XX` and `YY` ? Express your answers as a function of n. (Note that n as shown in lines 1,2 is not tight for x, y! You can do much better that that!)

(b) Assuming a cost of 1 for `RaiseToPower(x,y)` and given (a) what would the running time of `PerfectPower` be for the optimal choices of `XX,YY` as they can be obtained through (a)?

(c) Incorporate into the result of (b) the running time of an efficient implementation of `RaiseToPower(x,y)`. (Use the running time only. You don't need to show the algorithm itself. A non-efficient implementation is one than involves the product of y xs as in $x * x \ldots x$ and takes $y - 1$ multiplications!)

(d) Have you been able to find a solution through (a)-(c) whose running time is $O(n^a \cdot \lg^b n \cdot \lg \lg^c n)$ where $a < 1$, $b > 0$, $c > 0$ are constant? Explain by providing the values for the constants a, b, c established.

(e) Can you obtain a solution for the `PerfectPower(n)` problem whose running time is $O(\lg^b n \cdot \lg \lg^c n)$ where b is kept around 2 and c around 1 or 2 (and both are constant)? Explain. (You can modify extensively `PerfectPower` or provide a succinct English/pseudocode description for that. The choice is yours.)

Hint. Think binary.

Part (f) is optional.

(f) How far can you go in solving this problem? Can you achieve an algorithm whose running time is $O(\lg n \cdot \lg \lg^c n)$, for some constant $c \leq 3$? Explain.

Problem 7. (50 POINTS)
Implement an algorithm for the problem in Problem 6 whose worst-case running time is $O(\lg^k n)$ for any constant k of your choice. (You need to be able to solve Problem 6 for parts (a) through (e).)

The deliverable part of this problem would be a function or class `perfectp` that through the command line a user can interact and determine whether integers input in the command line are perfect powers or not!

```
%  ./perfectp  16                                  // A C or C++ invocation
16 is a perfect power since 4^2 = 16               // One possible output

%  ./perfectp  48
48 is not a perfect power

%  ./perfectp  887503681
887503681 is a perfect power since 31^6 = 887503681

%  java perfectp 16                                // A java invocation
16 is a perfect power since 2^4 = 16               //

%  ./perfectp 1860867
1860867 is a a perfect power since 123^3 = 1860867
%  java perfectp 4294967296
4294967296 is a a perfect power since 2^32 = 4294967296
```

It is very likely that Java might have a class for doing this. You are not allowed to use it. Your code will be inspected for that.

The expectation for a "correct" implementation (i.e. one that adheres to Problem 6(a)-(e)) is to run each one of the cases above in sub 15second running times in C, C++ or Java. (Thus a direct implementation of the 11 steps of `PerfectPower` as shown in Problem 6, will not achieve this.)

Problem 8. (100 POINTS)
Implement in C, C++, Java the following four implementations of quicksort,
`qsf, qsl, qsm, qsr` that pick the splitter to always be the left-most (first), right-most (last), middle keys, or a uniformly at random key of the input.
Test your 4 implementations of 4 different problem sizes ($n = 64000, 256000, 1024000, 8192000$), and 4 different input sets of `double` keys (as in real numbers, not integers!). The 4 different inputs are a random sequence of doubles in the range 0 to $n-1$ or so, all keys are the same and equal to 123.123, a decreasing sequence $n, n-1, \ldots, 2.0, 1.0$, and an increasing sequence $1.0, 2.0, 3.0, \ldots, n$.

One could generate those sequences as in

```
        for(i=0;i<n;i++)
          A[i] = (double) (n*(random()/((double)INT_MAX)));
          // rand(void) might be used instead of random()
          // RAND_MAX might replace INT_MAX as well in some
          // compilers
/*
          A[i]=  123.123;
          A[i]=(double) (n -i);
          A[i]=(double) (i+1);
*/
```

Time all you experiments and tabulate in a table submitted with your code. I also expect a minimal interface that for C/C++ would invoke through a call such as

`./hw1sort 1 2 3`

a selected algorithm (1 means qsf, 2 qsl, etc), a problem size (2 means $n = 256000$) and input (3 means decreasing sequence $n, n-1, \ldots$) that will time the sorting operation only (not the assignment of data) and report the wall-clock time accordingly. One way to obtain time information (at least in C/C++) is to use something like

```
double t1, t2;
 t1= clock();
   qsl(A,n);
 t2= clock();
 printf("Time is %f\n",(t2-t1)/((double)CLOCKS_PER_SEC));
 // CLOCKS_PER_SEC might cause problems in some compilers!
```