CS 667 : Homework 1(Due: Feb 14, 2012)

This Homework is Problems 1-5 and worth 200 points. You can replace some of these points with Problem 6 or 7, but you can only submit 200 points worth of problems.

Problem 1. (40 points)
(a) We have 4 coins all of the same weight except one that is fake and may weigh LESS or MORE than the other coins (we don't know what the case is). We also have a balance scale; any number of coins can be put on one or the other side of the scale at any one time and the scale will tell us whether the two sides weigh the same or which side weighs more (or less). If two coins are weighed (one on each side of the scale) and they do not weigh the same, we can't decide by this weighing alone which of the two is the fake. Can you find the fake coin with only 2 weighings? Explain (and justify your answer).
(b) You are given 27 coins, one of which is a fake and we now know that it is lighter than the rest. What is the minimum number of weighings to determine the fake coin? Prove your arguments.

Problem 2. (40 points)
You are given n arbitrary integers in the range 0 to $m-1$. Process the data in such a way that queries of the form "How many of the n integers are in range $[a,b]$ for given a,b" can be answered in $O(1)$ time. Your algorithm (processing part) should take no more that $\Theta(n+m)$. After this preprocessing you should be able to answer queries in $O(1)$ time.

Problem 3. (40 points)
Professor I. M. Nuts of the Math Department is scratching his head over the following problem. Given two sequences of numbers $X = \langle x_1, \ldots, x_n\rangle$ and $Y = \langle y_1, \ldots, y_n\rangle$ he must find the $m \leq \sqrt{n}$ largest values of the sums $x_i + y_j$, $1 \leq i, j \leq n$ in SORTED ORDER (smallest to largest). One may assume that $m \leq \sqrt{n}$ but one must not assume that m is a constant. The proposed algorithm should have worst-case running time that is $o(n^2)$. He requests the assistance of the CS Department in dealing with this problem.

(a) Give an algorithm whose worst-case running time is $o(n^2)$ that solves this problem; an $\Omega(n^2)$ time algorithm will not get you any points.

(b) Does there exist an algorithm whose worst-case performance is $\Theta(n)$? Explain, i.e. show/prove that such an algorithm does not exist or give the algorithm and analyze its performance, if such an algorithm exists.
Example. If $X = \langle 2, 0, -2, -3, 1, 100, 290, 300, 4\rangle$ and $Y = \langle -20, -19, -8, -200, 11, 1, 10, -7, 0\rangle$ and $m = 3$ then the answer is $\langle 301, 310, 311\rangle$. Note that 301 can result either as the sum of $290 + 11$ or $300 + 1$.

Problem 4. (40 points)
An arbitrary sequence $S = \langle s_1, s_2, \ldots, s_n\rangle$ of $n > 1$ positive integers is given. The key values are otherwise arbitrary and no assumption about their size can be made. Let $D = Max(S) - Min(S)$, where $Max(S)$, $Min(S)$ are the largest and smallest integers in S respectively.

Given S, give an algorithm that finds two integers a, b in S such that $a \geq b$ and $(a-b) \leq D/(n-1)$ in **linear** (i.e. $O(n)$) worst-case running time.
Example. Let $S = \langle 1, 900, 840, 135, 2001\rangle$. Then $D = 2001 - 1 = 2000$ and $D/(n-1) = 2000/4 = 500$. A solution to the problem is $a = 135$, and $b = 1$. There is another solution as well: $a = 900$ and $b = 840$.
Note. It suffices to find a single pair (a, b); you don't need to find all possible pairs nor the best such pair.

Problem 5. (40 points)
We have an array A[0..n-1] of n keys that have at most k distinct values, where $k \leq \sqrt{n}$. (The fact that k is bounded is known in advance.) We intend to sort this array in time faster than the generic $\Omega(n \lg n)$ depending on the value of k, by taking into consideration that there are not many distinct values among the n keys. We will do so in two rounds of computation. In the first round, we will compute a sorted array X[0..k-1] that contains the k distinct keys of A. Then, in the second round we can sort A using X as a guidance. You are asked to implement the first step of this algorithm.

Design an algorithm that given A, k and n as input determines the $k \leq \sqrt{n}$ distinct keys of the input, and stores them in array X in sorted order without using more than $\Theta(1)$ additional extra space. Your algorithm should run in worst-case running time of $O(n \lg k)$. Note once again that $k \leq \sqrt{n}$. Briefly explain the algorithm, comment on its correctness, and analyze its worst-case running time.
Observation. $O(n \lg k)$ is $O(n)$ if k is constant; if $k = O(\lg n)$ then $O(n \lg k) = O(n \lg \lg n)$, which is $o(n \lg n)$.

Problem 6. (40 POINTS)
Implement in C, C++, or Java the following four implementations/variant of quicksort available in the textbook (Chapter 7, page 171 for CLRS3e, or Chapter 7, page 146 for CLRS2e). (The guidance code in the textbook implements one of them anyway; you need to modify THIS code to get the other three, but you are not allowed to change the structure of quicksort.)
These are `qsf, qsl, qsm, qsr` that pick the splitter to always be the left-most (first for `qsf`), right-most (last, `qsl`), middle keys (`qsm`), or a uniformly at random key of the input (`qsr`).
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 32.25 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]=  32.25;
          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`

or

`java 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!
```

Problem 7. (80 POINTS)

Implement a `Select(i,n)` function for returning the i-th smallest of n keys in three different ways as described below.

- Sort the keys using quicksort (or the system's `qsort` function for C or C++, and `Array.sort` for Java), and then return the (index of the) i-th key of the sorted sequence.
- Use the Randomized Selection algorithm as described in the textbook as `qSelect` (page 216 or 185).
- Use the worst case linear time selection algorithm (page 220 or 189), using groups of 5 (det5select) and extend it to work also for groups of 7 (det7select); the former is described in the textbook, infer the latter from the former.

Thus provide the implementation of the following four functions respectively.

```
int  sortselect(double keys, int n, int stat );  // C/C++ shown
int  qselect    (double keys, int n, int stat ); // Adjust accordingly interface for Java
int  det5select (double keys, int n, int stat );
int  det7select (double keys, int n, int stat );
```

Note that the value returned is an integer index (0 to $n-1$), not a key value. Also, `stat` is a value between 1 and n; the former stat value indicates the minimum (first-order statistic) and the latter the maximum (n-th order statistic). However the index returned is between 0 and $n-1$ (inclusive of the end-points).

Test your 4 implementations on 3 different problem sizes ($n = 1024000, 4096000, 16384000$) on random inputs (see the previous problem) and tabulate timing results in a 4×3 table. Note that the input must be the same for each of the 5 runs for a given problem size. You need to provide a minimal interface (see below for C/C++ and Java) that accepts one command-line argument that is an integer from 1 to 4 (the 4 below indicates `det7select`, another command-line argument which is the problem size and a last one which is the statistic. A side-effect of this program `hw1select` is the dumping into file `hw1dump.txt` of the n keys, one value per line.

```
./hw1select 4 n stat
```

or

```
java hw1select 4 n stat
```

IF YOU PLAN TO SUBMIT PROBLEM 7 or PROBLEM 8, READ HANDOUT 2 first. Tabulations or experimental results are in the form of comments embedded in the code as instructed in Handout 2.