CS 584/684 Spring 2017 Homework 4 – due noon, Wednesday, May 3 2017

Your solutions to problems 1 and 2 should be type-set in LaTeX and submitted in both `.tex` and `.pdf` form, with file names `hw4.tex` and `hw4.pdf`. These two files, plus any additional source files invoked from your `.tex` file (such as pictures), should be bundled together into a single `.zip` file named *your-last-name*`-tex-hw4.zip`. Your code for problem 3 should be submitted in a single, separate `.zip` file named *your-last-name*`-code-hw4.zip` with contents as described below in the description of problem 3.

Submit by emailing to `hamialex@pdx.edu` including the zip file as a separate attachment and including "`CS584 HW4`" in the subject line.

All algorithms must be accompanied by proofs of correctness and of running time.

1. Treaps.

(a) Prove that for any collection of (key,priority) pairs such that the priorities are all distinct and the keys are all distinct, the treap having those pairs as internal nodes is uniquely defined.

(b) Consider the following standard code for inserting a value directly into a functional BST (without regard to balance):

```
TINSERT(t, k)
1  if t = LEAF
2       return NODE(LEAF, k, LEAF)
3  else let (t_l, k', t_r) = t
4       if k < k'
5            return NODE(TINSERT(t_l, k), k', t_r)
6       elseif k > k'
7            return NODE(t_l, k', TINSERT(t_r, k))
8       else // k = k'
9            return t
```

Suppose we start with an empty BST and insert n keys $k_n, \ldots, k_1$ in that order by repeatedly calling TINSERT. Prove that the resulting tree is the same as we would get by taking the treap (unique by part (a)) with (key,priority) pairs $(k_1, 1), (k_2, 2), \ldots, (k_n, n)$, and erasing the priorities from the nodes.

(c) Assuming that the inputs to SPLIT and JOIN (as defined in the lecture notes) are indeed treaps, prove that the outputs are too, i.e. that they have both the BST property and the heap property.

(d) Describe a parallel divide and conquer algorithm for computing the *union* of two treaps, i.e., a treap whose keys are the union of the keys in the input treaps. Your algorithm should use SPLIT and JOIN as primitives. Give *rough* bounds on the work and span of your algorithm, as a function of the sizes of the input treaps.

2. Insertion sort.

(a) An *inversion* in an array of distinct integers is a pair (i, j) such that $i < j$ but $A[i] > A[j]$. Use indicator random variables to compute the estimated number of inversions in a randomly-ordered array.

(b) Consider the INSERTION-SORT procedure on CLRS p. 18. Compute the asymptotic *average-case* running time of this procedure on an input array of n distinct elements, assuming all permutations of the input values are equally likely. Hint: First, relate the running time of INSERTION-SORT to the number of inversions in the input array.

3. Hashing.

Implement the randomized hashing scheme described in the lecture notes, using the hash function

$$h_a(x) = \left\lfloor \frac{(a \cdot x) \bmod 2^w}{2^{w-l}} \right\rfloor$$

using the fixed value $w = 32$. However, rather than picking the salt a at random, you will take a as an input to your main program, along with the values to be stored in the hash table. The output of the program will report a histogram of how many input values collide.

Note that although this program can certainly be implemented in any programming language, the computation of the hash function will be easiest in a language that supports 32-bit unsigned integers as a native type.

You may assume that programs will be tested on a machine with at least 4GB of memory.

Your program should take one command line argument, which is the name of an input file. The format of that file will be as follows:

- First line contains a number $C \geq 0$ of test cases in the file.
- Then come C test cases, each of the following form:
  - Line containing the salt a, where $1 \leq a \leq 2^{32} - 1$ and a is odd.
  - Line containing number l, where $0 \leq l \leq 32$ and $m = 2^l$ is the size of the hash table.
  - Line containing number n, the number of values being hashed into the table, where $0 \leq n \leq 10^6$.
  - Line containing the n values to be hashed into the table, with exactly one space separation between each value. The values are distinct. Each value is an integer in the range $[0 \,..\, 2^{32} - 1]$.

Your program should output (to `stdout`) one line for each test case, of the form "`Case i:` c_1 z_1 c_2 $z_2 \ldots c_k$ z_k" where the pair c_i z_i means that there exactly z_i slots for which the number of values that hashed to the slot was c_i. Only pairs with $c_i > 0$ should be reported. The pairs should be reported in *decreasing* order of c_i. There should be one space separation between each number on the line.

Example Input:

```
2
1
3
5
1 2 3 4 5
1
32
5
1 2 3 4 5
```

Corresponding Example Output:

```
Case 1: 5 1
Case 2: 1 5
```

Warning: If your output format is not correct (even spacing), you will get no credit; this problem will be graded by doing a `diff` against a standard output file.

Place your code (one or more source files) together with a `Makefile` in a fresh subdirectory with the name *your-last-name*`-code-hw4`. Then create a single `zip` archive with the name *your-last-name*`-code-hw4.zip` containing just that directory and its contents. It should be possible to build an executable file called `hw4` and test it on an input file `/path/to/foo` by the following steps:

1. `unzip` *your-last-name*`-code-hw4.zip`
2. `cd` *your-last-name*`-code-hw4`
3. `make`
4. `./hw4 /path/to/foo`