Homework 8
Out: 5 Apr. Due: 12 Apr.

Instructions: Put your solutions in the homework box on Soda level 2 by 5pm on Thursday. Take time to write clear and concise answers; confused and long-winded solutions will be penalized. You are encouraged to form small groups (two to four people) to work through the homework, but you must write up all your solutions on your own. Depending on grading resources, we reserve the right to grade a random subset of the problems and check off the rest; so you are advised to attempt all problems.

1. MU, Exercise 13.2. [NOTE: In part (b), you should just give the construction; you do not need to provide a proof that it works.]

2. Consider the problem of deciding whether two integer multisets $S_1$ and $S_2$ are identical (in a multiset, each element can appear multiple times) in the sense that each integer occurs the same number of times in both sets. This problem can obviously be solved by sorting in $O(n \log n)$, where $n$ is the cardinality of the multisets. In this problem we will consider a more efficient randomized algorithm based on hashing. Here is the algorithm:

   - Hash each element of $S_1$ into a hash table with $cn$ counters (where $c > 1$ is a constant), using some 2-universal family of hash functions. The counters are initially 0, and the $i$th counter is incremented each time the hash value of an element is $i$. Using another table of the same size and the same hash function, do the same for $S_2$.
   - If the $i$th counter in the first table matches the $i$th counter in the second table for all $i$, output “yes”; otherwise, output “no”.

(a) What is the running time of this algorithm? Assume that hashing and arithmetic operations (for incrementing and comparing counters) take constant time.

(b) Verify that if $S_1$ and $S_2$ are identical, the algorithm outputs “yes” with probability 1.

(c) Show that if $S_1$ and $S_2$ are not identical, the algorithm outputs “no” with probability at least $1 - \frac{1}{c}$.

   You may assume for simplicity that $S_1$ and $S_2$ are disjoint (indeed, this is WLOG because we can remove elements common to $S_1$ and $S_2$ in the analysis). [HINT: suppose $S_1$ and $S_2$ are disjoint. Fix some element $x \in S_1$. Now show that with probability $1 - \frac{1}{c}$ over $h$, the $h(x)$th counter for $S_2$ is 0.]

3. Let $L$ be a language (which we can think of as a decision problem, where “yes” instances correspond to strings $x \in L$ and “no” instances to strings $x \notin L$). Suppose we have a randomized algorithm $A$ for $L$ with one-sided error; i.e., on any input $x$,

   (i) if $x \in L$ then $A(x)$ outputs “yes” with probability at least $\frac{1}{2}$;
   (ii) if $x \notin L$ then $A(x)$ outputs “no” with probability 1.

   As we know very well, we can reduce the error probability by performing repeated independent trials of $A$; to get the error probability down to $\delta$, we need $\lceil \log_2(\delta^{-1}) \rceil$ trials, which requires $O(t \log(\delta^{-1}))$ random bits. In this problem we will see how to use pairwise independence to achieve error probability $\delta$ using only $O(t)$ random bits (for any $\delta \geq 2^{-t}$).

   To do this, it is helpful to think of $A$ as taking two inputs, namely $x$ and a string $r$ of random bits of length $t$, where $t$ is the running time of $A$ on $x$. Then the above properties translate to the following:

   (i) if $x \in L$ then $A(x, r)$ outputs “yes” for at least half of the strings $r$;
   (ii) if $x \notin L$ then $A$ outputs “no” for all strings $r$. 


(a) Suppose now that we pick \( s \) pairwise independent uniform random strings \( r_1, \ldots, r_s \in \{0, 1\}^t \), and output “yes” if at least one of the trials \( A(x, r_i) \) outputs “yes”. Show that the error probability for this algorithm is \( \frac{1}{s} \). [HINT: Let \( Y = \sum_{i=1}^{s} Y_i \), where \( Y_i \) is the indicator r.v. for the event that \( A(x, r_i) \) outputs “yes”. Show that \( \text{Var}[Y] \leq \frac{s}{4} \), and use Chebyshev’s inequality.]

(b) Explain briefly why only \( O(t) \) random bits are needed to implement the scheme in part (a). Note that the number of random bits needed is independent of \( \delta \).

(c) What is the running time of this scheme (as a function of \( t \) and \( \delta \)), and how does it compare to the standard approach based on independent trials? Ignore the time taken to generate pairwise independent random strings.

4. In this problem, you are asked to run some simple experiments that illustrate the “power of two choices” for balls and bins, as discussed in class.

(a) Write a simple program that simulates the balls and bins experiment for given values of \( n, m \) up to 1 million, and outputs the number of balls in the fullest bin. [NOTES: You will need a random number generator. The recommended one on most Unix implementations is \textit{drand48}; see the man page for details. In any case you should state which random number generator you used. Also, it is wasteful of space to use 1 million integers to store the bin loads. Since you are very unlikely ever to see a load greater than 15, you can in fact use a single byte (i.e., a character) to store the loads.]

(b) Perform 100 simulations with \( m = n = 10^6 \) and draw up a table of the maximum loads you observe.

(c) Now consider the following alternative “two choices” scheme: balls are again thrown sequentially, but instead of simply choosing a single bin at random, each ball now chooses \textit{two} bins at random, inspects their current loads, and goes to the least full of the two. (If both loads are the same, the ball can decide arbitrarily.) Modify your program to implement this scheme. Again, perform 100 simulations with \( m = n = 10^6 \) and tabulate the maximum loads you observe. Compare your results with those of part (b).

(d) Finally, modify your program again to allow each ball three choices rather than two. Repeat the above simulations and compare your results again.