Instructions:

Pick any two of the following problems to do and turn in solutions for those. Do not turn
in solutions to more than 2 problems (or we will arbitrarily pick 2 and grade only those.) The
theoretical problems listed below are roughly in order of increasing difficulty.

Unless specifically forbidden, you are allowed to collaborate with fellow students taking the
class in solving problem sets, however you must always write the solutions up on your own. If you
do collaborate on solving the problems, you must acknowledge for each problem the people you
discussed that problem with (other than the TA and instructor).

The problems have been carefully chosen for their pedagogical value and hence might be similar
or identical to those given out in past offerings of this course at UW, or similar courses at other
schools. Using any pre-existing solutions from these sources, or any source not explicitly linked to
or discussed on our course web page constitutes a violation of the academic integrity expected of
you and is strictly prohibited.

Concise, clear and precise solutions are appreciated (and expected).

1. In Matlab or another suitable programming environment, implement two of the possible uni-
versal hash functions described in one of the following: http://courses.cs.washington.
edu/courses/cse521/15sp/refs/thorup1.pdf or http://www.cs.cmu.edu/afs/cs/project/
pscico-guyb/realworld/www/slidesS14/hashing.pdf Use these two hash functions to
map \{100, 200, 300, \ldots, 100n\} to a table of size around \(n\) (with open chaining). It would be
great if one of them was the hash function described in Section 2.3 of the first reference, but

not required. (Use \(n = 10^6\) for starters, but you can play around with different values of \(n\), as
you like.) For each of the randomly selected hash functions (one from each universal class),
report:

- the time it takes to hash all the elements,
- the length of the largest bucket,
- the sum of the squared bucket sizes, i.e. \(\sum_{i=0}^{m-1} n_i^2\), where \(n_i\) is the number of keys
  hashing to location \(i\).

Repeat the experiment at least 3 times, choosing a new random hash function from the
universal class each time. For comparison, measure these same quantities for your favorite
hash function (either whatever you normally use, make up something or find something). See
if you can find another set of \(n\) keys that will result in a lot of collisions for your favorite hash
function, and repeat the whole experiment. Write up what you did, what results you got and
give a brief discussion of the results. Include a brief description of any design decisions.
Include a copy of your code.

2. Repeat the entire experiment described in the previous question, but use linear probing
instead of chaining. This time the numbers you will report are the time it takes to hash all the elements, the longest probe sequence you
encounter, and the sum of the squares of the probe sequence lengths. Compare the results to those you get if you use a completely random hash function. Run the experiment with table sizes of $2^n, 3^n, 4^n$ and $5^n$.

3. Consider a Bloom filter where instead of hashing each item $k$ times into the same table of size $m = cn$, the table is divided into $k$ equal size pieces (of size $m/k$) and each of the $k$ hash functions maps the elements to a different piece. As usual, on insertion of an element $x$, each of the locations $h_i(x)$ is set to 1.

What is the probability of a false positive? Which would you prefer to use between this kind of Bloom filter and the original one and why? (Note that for $0 < x < 1$ and $y \geq 2$ integer we have $(1 - x)^y > (1 - xy)$.)

4. Write out the full details of the analysis I showed in class, that for uniform, independent random hashing with linear probing, the expected time to insert an element into a table of size $2n$ containing less than $n$ elements is $O(1)$. You may find the following facts useful: For $0 < x < 1$,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \quad \text{and, by differentiation,} \quad \sum_{k=1}^{\infty} kx^{k-1} = \frac{1}{(1 - x)^2}.$$

There is a high level discussion of this analysis in this video [http://www.di.ku.dk/summer-school-2014/course-material/rasmus-pagh/](http://www.di.ku.dk/summer-school-2014/course-material/rasmus-pagh/) (see video for first lecture called “Basics of hashing: $k$-independence and the impact on applications”, starting roughly at minute 37:37).

5. Consider the following process for matching $n$ jobs to $n$ processors. In each step, every job picks a processor at random. The jobs that have no contention on the processors they picked get executed, and all other jobs back off and then try again. Jobs only take one round of time to execute, so in every round all the processors are available.

- Suppose that there are $r$ jobs left at the beginning of some round. What is the expected number of jobs remaining to be completed after that round?
- Suppose that in each round, the number of jobs completed is exactly equal to its expectation. Show that (under this false assumption) the number of rounds until all jobs are finished is $O(\log \log n)$. You may want to use the fact that for $0 < x < 1$ and integer $k \geq 0$, $(1 - x)^k \geq 1 - kx$. (Although the assumption is false, the conclusion is true.)
- **Extra credit:** Prove (without the erroneous assumption) that the expected number of rounds until all jobs are finished is $O(\log \log n)$.

6. Write out the full details of the analysis showing that for hashing with linear probing, using a random hash function from a 6-wise independent family of hash functions $\mathcal{H}$ the expected time to insert an element into a table of size $m = 2n$ containing less than $n$ elements is $O(1)$. Here $\mathcal{H}$ has the property that for all $x_1, \ldots, x_6 \in U$, and all $y_1, \ldots, y_6 \in [m]$,

$$Pr(h(x_1) = y_1, \ldots, h(x_6) = y_6) \leq m^{-6}.$$

There is a high level discussion of this analysis in this video [http://www.di.ku.dk/summer-school-2014/course-material/rasmus-pagh/](http://www.di.ku.dk/summer-school-2014/course-material/rasmus-pagh/) (see video for first lecture called “Basics of hashing: $k$-independence and the impact on applications”, starting roughly at minute 47:44).
I've taken a couple of these problems from http://www.cs.princeton.edu/courses/archive/fall14/cos521/homeworks/hw1.pdf