STA 247 — Assignment #2, Due in class on November 24 at 3:10pm

Late assignments will be accepted only with a valid medical or other excuse.
This assignment is to be done by each student individually.
There are three pages in this assignment

Part I

Imagine a university computer server that processes two types of requests. One type of request asks which courses are being taken by a particular student. These requests require accessing 10 records from a database. The other type of request asks which students are taking a particular course. These requests require accessing 20 records from a database. The database caches records that it thinks might be accessed in the near future, so that only some of these accesses will require actually reading data from disk. (For instance, the last 1000 records that were accessed might be kept in the cache, if it is thought that these are more likely to be accessed again, but other strategies are possible as well.)

Suppose that 80% of all requests are of the first type, which ask for the courses taken by a particular student, and 20% are of the second type, which ask for the students taking a particular course. Assume that the type of a request is independent of the types of other requests.

Suppose also that the database's caching strategy is good enough that only 30% of the database records that are accessed have to be read from the disk containing the data. (The other 70% are in the cache, and can be delivered quickly without reading data from the disk.) Assume that whenever a record is accessed, whether this record is in the cache or not is independent of whether any of the other records accessed were in the cache.

1. What is the distribution for the number of records that have to be read from disk (ie, that aren't in the cache) when a request of the first type is made? Find the mean and standard deviation of this distribution, and find the probability that more than 5 records will have to be read from disk to satisfy this request.

2. Find the mean and standard deviation of the number of records read from disk in order to process a request that may be of either type (with the types having the probabilities given above). Use Chebychev's inequality to give as small an upper bound as you can for the probability that *more than* 7 records will have to be read from disk to satisfy this request. Compare this upper bound with the exact value, which you should compute to at least four decimal places of accuracy.

3. Suppose that the server receives 50 requests every minute. Find the mean and standard deviation of the number of disk accesses it makes to process these 50 requests.

4. Discuss whether or not it is realistic to assume, as stated above, that whether one record accessed is in the cache is independent of whether any other records accessed were in the cache. Why might this not be true? What might be the effect on the means and standard deviations you found above if this assumption is not true (but the other assumptions stated above are still true)?

You should find the answers to these questions by hand (not by doing a computer simulation), except that you may use the R expression `dbinom(i,n,p)` to compute the probability that a binomial random variable will have value `i`, and the expression `pbinom(i,n,p)` to compute the probability that a binomial random variable will have value less than or equal to `i`.

Part II

In class, we discussed one way to reduce the number of entries that have to be searched when looking for a symbol — using a hash table, so that we only need to search the list of symbols in one "bucket". Another approach is to keep all the symbols in one list, but to try to arrange that the symbol that is being looked for is likely to be at the front of the list. For this assignment, you will evaluate how well two schemes of this sort work, using various probabilistic models of how likely the various symbols are to be accessed.

Both schemes keep an ordered list of symbols, and search for a symbol by looking through this list in order, until the symbol is found (or until the end of the list, if the symbol isn't present). Once a symbol is found, the first "move to front" scheme moves this symbol to the front of the list, moving the symbols before it back to make room. The second "move one forward" scheme moves the symbol that was found just one place closer to the front, by swapping it with the symbol before it in the list. For both schemes, if the symbol found is already at the front of the list, the list is left unchanged.

For example, suppose that the table initially contains the following symbols:

```
FRED   MARY   GEORGE   PAUL   ALBERT   HANNA
```

If we now try to look up the symbol `PAUL`, both methods will need to compare this symbol with `FRED`, `MARY`, `GEORGE`, and finally with `PAUL`, at which point the symbol will be found. After finding `PAUL`, the "move to front" scheme will change the order of symbols in the table to the following:

```
PAUL   FRED   MARY   GEORGE   ALBERT   HANNA
```

If we had used the "move one forward" scheme instead, the order of symbols would have changed to the following:

```
FRED   MARY   PAUL   GEORGE   ALBERT   HANNA
```

When evaluating which scheme is better, we needn't store actual symbols. Instead, we'll store positive integers, which will be faster and easier. We'll initialize the table to contain the integers from 1 to `table.size` (which you should set to 100 for this assignment), in a random order. We'll then try looking up integers in this range in the table, which will cause the order of integers to change, as controlled by whichever scheme we're using. We'll record the position in the list where each of the integers we look up was found, which will be roughly proportional to the amount of computer time required to find it.

How much time each scheme takes will depend both on what symbols are looked up, and on the order in which they are looked up. We can use a probability distribution for symbols to model a situation in which some symbols are more likely to be looked up than others. To model the effects of order we need to say how the symbol looked up may depend on the symbols looked up earlier.

We will consider two distributions for symbols, which we are representing by integers from 1 to `table.size`. The first is uniform, in which each symbol has probability 1/`table.size`. You can produce a vector with such probabilities with the R expression `rep(1/table.size,table.size)`. The second distributions follows what is called "Zipf's law", in which symbol `i` has probability proportional to `i/table.size`. This "law" has been found to be a good approximation to the frequencies of many real symbol sets, such as words in the English language (though it's not really a "law" in the mathematical sense). You can produe a vector of probabilities that follow Zipf's law with the R expression `(1/(1:table.size)) / sum(1/(1:table.size))`.

In many situations, a symbol that was looked up recently has an increased probability of being looked up again. To model this, with probability p, the next symbol looked up will be one of the previous ten symbols looked up, with equal probabilities for each of these ten symbols (except that if a symbol was looked up more than once in the last ten lookups, it gets more than one chance). Otherwise (ie, with probability $1 - p$), we choose a symbol at random from the symbol distribution being used (either the uniform distribution or the Zipf's law distribution). The first ten lookups are always of symbols chosen from the distribution being used. For this assignment, we will consider only the values $p = 0$ and $p = 1/2$ for this dependence parameter. When $p = 0$ the symbol looked up will not actually depend on which symbols were looked up recently, whereas when $p = 1/2$, a recent symbol is looked up again about half the time.

You should evaluate how long it takes on average to look up a symbol for each of the two schemes ("move to front" and "move one forward"), using each of the two symbol distributons (uniform and Zipf's law), and the two values $p = 0$ and $p = 1/2$ for the dependence parameter.

To get you started, I have provided (on the course web page) an R function called `mtf.lookup`, which implements the "move to front" scheme, returning as its value the position of the symbol found (before it was moved to the front). You should write a similar function called `m1f.lookup` that implements the "move one forward" scheme.

You should also write a function called `simulate.lookups`, which simulates some number of lookup operations using one of these schemes, using the model for how symbols are chosen that is described above, with specified probabilities for symbols and a specified probability of looking for the same symbol as one of the last ten lookups. In detail, the `simulate.lookups` function should take five arguments, as follows:

`lookup`	Function used to look up a symbol in the table, either `mtf.lookup` or `m1f.lookup`.
`table.size`	Size of the table. Should be set to 100 for this assignment.
`sym.probs`	Vector of symbol probabilities (of length `table.size`).
`last10.prob`	Probability that the symbol to look up is picked from the last 10 looked up (p above).
`n.lookups`	Number of symbol lookups to simulate. Set this to 8000 for this assignment.

The `simulate.lookups` functions should start by creating a table stored in the global variable `sym.table`, containing the integers from 1 to `table.size`, in a random order. A random permutation of these integers can be obtained with the R expression `sample(1:table.size)`. The `simulate.lookups` function should then simulate `n.lookups` lookup operations, using the `lookup` function that was passed as the first argument. The results of each lookup (the position of the symbol found) should be saved in a vector. This vector should be the value returned by `simulate.lookups` once all the lookups have been done.

You should call `simulate.lookups` with the eight combinations of values for the `lookup`, `sym.probs`, and `last10.prob` mentioned above. For each such combination, you should save the vector returned by `simulate.lookups` in a variable, say `res`, and then use it to do the following:

1) Plot a graph showing the positions at which the symbols were found, using the command `plot(res)`.

2) Compute the sample mean and sample standard deviation of these positions, using the `mean`, `var`, and `sqrt` functions.

3) Compute the sample mean and sample standard deviation of only the last 4000 of these positions. You can get the last 4000 as `res[4001:8000]`.

You should then do each of the simulations twice more, and for each compute the sample means for all positions and for the last 4000 positions, as above. (You needn't hand in any output for these repeat simulations.) Comment on how variable the results are, and how this compares to what you would expect given the standard deviation of position that you found.

Finally, draw conclusions from your results about when the "move to front" scheme is better or worse than the "move one forward" scheme. Also comment on why you think the initial part of the simulation sometimes looks different from the remaining part, and what this says about the usefulness of the two schemes. Can you you think of a scheme that might sometimes be better than either of these schemes?

You should hand in (on paper) a listing of your functions, nicely formatted, with suitable but not excessive comments, along with items (1), (2), and (3) above for each of the eight combinations of argument values, and your comments on the results, as asked for above.