Problem

We need you to write a system that will process our email batches and decide whether or not to send each email based on the following rules:

1. Must send a maximum of one email per address.
2. Must never send an email with a spam score of more than 0.3.
3. The running mean of spam ratings must never get above 0.05. (In other words, as each email is processed the mean spam score of all the emails that have sent in the batch needs to be recalculated and can't ever be more than 0.05.)

The mean spam score of the most recent 100 emails sent can't go above 0.1.

The input is a collection of email records, each of which will have an `:email-address` key and a `:spam-score` key. We're not very good at math or counting, so we're not exactly sure how many emails there will be. Last time I think they said there were 5 million. Or maybe it was 10 million? Gotta remember to ask Randy about that later
Here's one example email record:

```
{:email-address "john123@dired.com"
 :spam-score 0.12}
```

Approach

The exercise was done in an iterative fashion trying to improve performance bottlenecks in different parts of the algorithm. The iterations and scratch-work can be found for reference in *emails.clj* . The final algorithm is in *final_emails.clj*

Running instruction:

lein repl

```
user=> (load-file "final_emails.clj")
#'user/process_email
user=> (ns emails)
emails=> (process_email 10000000)
emails=> (run-tests)
```

Code:

The function *ask_Randy* generates a list (of a given length) of random records of the format

```
{:email-address "john123@dired.com"
 :spam-score 0.12}
```

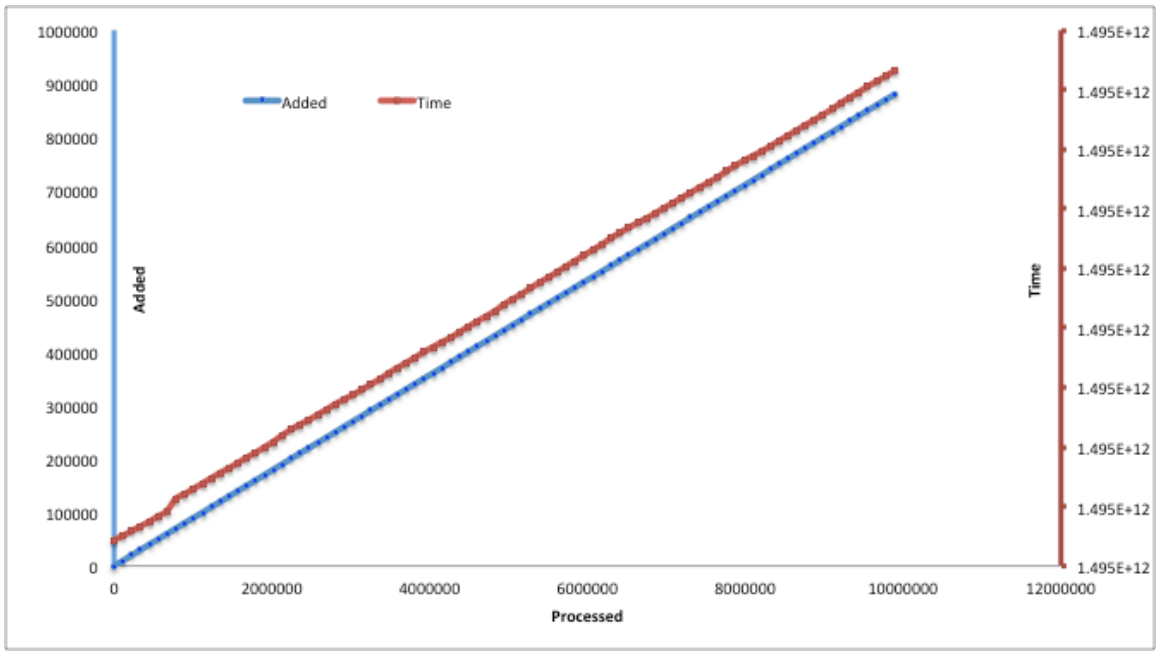
The function *process_email* first generates this email batch and then processes the batch using the four given rule.

Optimizations assumptions :

One of the major assumptions made in this algorithm (which heavily influenced the design) is that the number of duplicate emails in the records is far less than the number of actual

records. Since we can send only one email per address, I pre-processed the emails to create a list of duplicates. The alternative was to check whether each record has been added before or not. These two approaches would have similar worst-case performances but, if, however, the number of duplicates were far less than the length of the total collection, the former approach would perform better on average.

The batch processing time is linear to the length of the collection. Since the records are randomly generated the number of records added is linear with the number of records processed.



Benchmark performance:

Total running times for the full *process_email* method is

1 Million: 0.15 min
5 Million: 0.766 min
10 Million: 1.616 min