Matthew Koken
COEN 177 Lab 3

LRU

The deque (deck) holds all the pages. When a page is requested, if the
page is found, it is removed from the deck and pushed to the front (most recently used)
If it is not found in the deck, this is a page fault and it needs to be added.
If there is room, it can immediately be pushed to the front. If there is no room,
the back is popped (least recently used removed) and the page is pushed to the front. This keeps the order of use for all pages, keeping only the most recently used and purging those that are least recently used.

Second Chance (Like CLOCK)

This algorithm gives uses a reference bit to see if a page has been recently used. If there is room, the page will be added to the head with a reference bit of 1. If there is no room, the algorithm will look for an unreferenced page. It begins looking from the hand, marking any page it passes by to 0 unless it finds one already at 0. In that case, the page with a 0 reference bit will be removed and the new page will be added in its place. Pages start with no chances (a reference bit of 0), and are given a chance if it is referenced again after initial insertion.

LFU

The least frequently used algorithm keeps track of the usage of a page. When a page is referenced, the reference bit for it's number of uses is incremented. When the cache is full, the algorithm seeks out the page with the lowest number of uses to remove. When a page is referenced again, it is moved to the head and its number of uses is incremented. This keeps an order of most recently used. If two pages have the same number of uses, the least recently used will be purged.

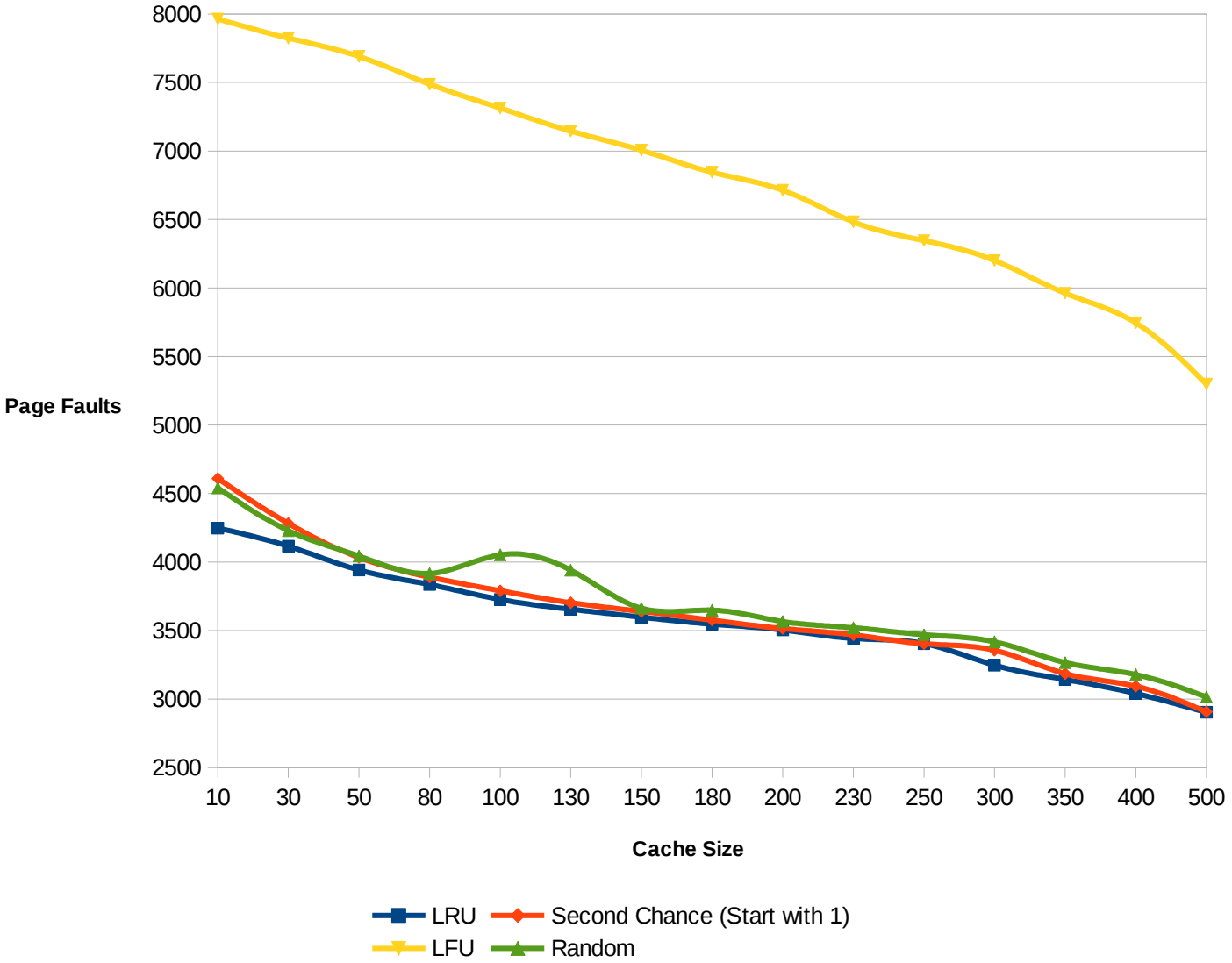
Random with Second Chance

This fourth algorithm behaves somewhat similarly to second chance. However, instead of incrementing the hand, the hand is turned to a random position within the cache. All pages are started with a reference bit of 1 as well. Testing for this will require multiple runs and averaging per cache size. THIS WILL NOT PERFORM THE SAME EVERY TIME BECAUSE OF THE RANDOMNESS.

Testing

In order to test the algorithms, different cache sizes will need to be used. A text file of 10000 integer entries, of which there are 2009 unique elements, is used as a simulation of page calls. Each page replacement algorithm will be tested with cache sizes varying from 10 to 500 pages in order to gauge the hit/miss rate. For random, multiple tests per each cache size will be needed to average the results.

Test Data



Cache Size	LRU	Second Chance (Start with 1)	LFU	Random
10	4248	4609	7963	4542
25	4116	4282	7824	4229
50	3942	4034	7690	4044
75	3836	3890	7488	3917
100	3727	3791	7313	4053
125	3656	3703	7145	3941
150	3598	3640	7005	3663
175	3547	3576	6845	3649
200	3505	3515	6714	3566
225	3443	3469	6483	3521
250	3405	3404	6346	3470
300	3248	3357	6201	3418
350	3143	3186	5962	3267
400	3041	3094	5746	3179
500	2904	2908	5300	3016

Results

Overall, LRU, Second Chance, and Random performed quite well. LRU had the overall lowest page fault rate, with Second Chance behaving slightly worse and Random performing similarly to Second Chance. LFU had quite high page fault rates, likely due to pages being used once and then not again for a while, so those get removed. Pages that are referenced many times then never touched again may not be removed for quite some time. Despite the randomness, Random performed quite well, and beat Second Chance in some cases with low cache sizes. Again, all algorithms were fed the same sequence of page requests, with the only variables being the algorithm and the cache size.

Improvements

For LRU, there don't seem to be many rooms for improvements in terms of hit rates. Using a better data structure for searching and deletion may be helpful for faster runtimes. Insertion only inserts at the head, so it would be quite fast in that aspect, and the same goes for deletion – only removing from the rear. The same could go for the other algorithms as well, as they are all implemented using the base of a deque for a data structure to hold pages. As long as page faults are kept to a minimum – the levels set by LRU – the best improvements will come from the operations that must be performed to handle the data.

For Second Chance, using a start from 1 reference could potentially lower page faults. If pages are being removed too quickly, ie they are not referenced again after insertion and a quickly deleted, then starting with a chance may keep the needed pages around long enough for them to be referenced again and avoid additional page faults.

For second chance random, some probability and statistics could improve our random guessing for a new place. If pages were to always be inserted towards the head, and moved to the head when reference again, a weight towards the end of the cache could provide a higher chance of finding an unreferenced page and reduce the number of new guesses. Otherwise, it would be unlikely to improve the performance much more than what Second Chance (CLOCK) is already capable of.

LFU however, is in the most need of improvement. It had quite high page fault rates. The best improvement I can think of would be to introduce some form of aging. As time goes by, maybe every clock cycle for example, or every new page request, the number of uses would be decremented. This would prevent pages that are requested multiple times and then never requested again from sitting in the cache and taking up space for long periods of time. By also giving a bias to recently used pages, less recently used pages that haven't been used many times should be removed first, leaving only the pages that are recently used frequently in memory.