INTRODUCTION

Before beginning this project, I read that the naive implementation of an LRU replacement policy (using a linked list that you pop the end off of when you need to add a new element and move a node to the front of on a cache hit) has an access time on the order of O(n).
Similarly, I read an article on geeksforgeeks.org (which I can no longer seem to find) that mentioned that adding and using a hash map to speed up access time could reduce access time to closer to O(1). I thought that would be an interesting claim to test.

To this end, I implemented both a Naive LRU replacement policy and an LRU policy similar to the one listed here (https://www.geeksforgeeks.org/lru-cache-implementation/) and ran some tests on each. Tests were run as follows:

TEST PROCEDURE

For testing purposes, I implemented my caches to simply store integers, but converting to store actual pages of data or page addresses should not drastically affect performance. Caches were limited to one of two sizes - a reasonably, but not trivially, small 100 entries, and a much larger 10000 entries. Randomly generated integers were used to test the caches.
For each cache size, I attempted to get results from differing percentages of cache hits; to do this, I tested each cache against accesses from a pool of integers 1.25 times the size of the cache (expecting mostly cache hits), a pool of integers 50 times the size of the cache (expecting mostly cache misses), and a pool 2.5 times the size of the cache (expecting around half cache hits, give or take about 10%). I assumed these groupings of potential accesses (henceforth called “cache entry ranges”) would give a good idea of how the cache performs under varied access conditions.
For each set of cache entry ranges, I also ran tests with a variable number of accesses. For the smaller caches, I tested on sets of 500 accesses to sets of 50000000 accesses, with a test at each order of magnitude in between. For the larger caches, I ran tests on the same benchmarks, excluding the tests where there would be fewer accesses than the size of the cache (I felt that it didn't make sense to test a cache removal policy when the cache would be sufficiently large to never have to remove a value).
Lastly, I ran 5 trials of each of the above tests so I could average out results and get a better idea of general performance.

TEST RESULTS

Let's start by discussing average total runtime. Below is a table of the average total runtime for all of the tests performed.

Average Total Runtime (Seconds)

	500 Accesses	5000 Accesses	50000 Accesses	500000 Accesses	5000000 Accesses	50000000 Accesses
LRU, Cache Size 100, Entry Range 0-125	0	0.0036	0.0302	0.2842	2.7326	26.9528
LRU, Cache Size 100, Entry Range 0-250	0.0006	0.0036	0.037	0.365	3.719	37.4486
LRU, Cache Size 100, Entry Range 0-5000	0.0008	0.004	0.0468	0.452	4.386	43.9714
LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	0.031	0.3106	3.1608	35.7972
LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	0.0478	0.4568	4.5226	51.948
LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	0.0512	0.485	4.5736	48.8762
Naive LRU, Cache Size 100, Entry Range 0-125	0.001	0.0086	0.0822	0.8184	7.9354	79.4544
Naive LRU, Cache Size 100, Entry Range 0-250	0.0006	0.0096	0.101	1.0014	10.1262	100.6496
Naive LRU, Cache Size 100, Entry Range 0-5000	0.0012	0.0118	0.1276	1.1826	12.1966	121.8502
Naive LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	5.9544	65.6238	655.4484	6935.658
Naive LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	8.2448	90.1862	912.645	9594.102
Naive LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	8.857	97.2602	989.2452	9678.922

Clearly the LRU outperforms the Naive LRU in average runtime. For a cache of 100 entries, barring the 500 access tests, which were so quick my timing code sometimes recorded less than a single tick of elapsed time, the Naive LRU tests took 2 to 3 times as long as LRU tests across all three cache entry ranges. For a cache of 10000 entries, Naive

LRU tests took two orders of magnitude longer to perform than their LRU counterparts, across all cache entry ranges (in particular, on the 10000 entry Naive LRU cache, the 50000000 access tests took more than 2 hours to run, whereas it took less than a minute to run the LRU equivalent tests).

The difference in runtime between the 100 and 10000 entry Naive LRU caches is also consistent with the kind of growth one would expect to see with an O(n) cost function – the cache grew by two orders of magnitude, and so did its test access times. Additionally, the LRU caches' total test runtime appear to be fairly consistent between the 100 entry and 10000 entry caches, with differences in runtime appearing to be more the result of having more accesses and differing cache entry ranges (and therefore, different predicted hit rates).

To verify these preliminary findings, we can also examine the average time per cache access; a table of those results is included below.

	Average Time Per Access (Seconds)					
	500 Accesses	5000 Accesses	50000 Accesses	500000 Accesses	5000000 Accesses	50000000 Accesses
LRU, Cache Size 100, Entry Range 0-125	0	7.2E-07	6.04E-07	5.684E-07	5.4652E-07	5.39056E-07
LRU, Cache Size 100, Entry Range 0-250	1.2E-06	7.2E-07	7.4E-07	7.3E-07	7.438E-07	7.48972E-07
LRU, Cache Size 100, Entry Range 0-5000	1.6E-06	8E-07	9.36E-07	9.04E-07	8.772E-07	8.79428E-07
LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	6.2E-07	6.212E-07	6.3216E-07	7.15944E-07
LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	9.56E-07	9.136E-07	9.0452E-07	1.03896E-06
LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	1.024E-06	9.7E-07	9.1472E-07	9.77524E-07
Naive LRU, Cache Size 100, Entry Range 0-125	2E-06	1.72E-06	1.644E-06	1.6368E-06	1.58708E-06	1.589088E-06
Naive LRU, Cache Size 100, Entry Range 0-250	1.2E-06	1.92E-06	2.02E-06	2.0028E-06	2.02524E-06	2.012992E-06
Naive LRU, Cache Size 100, Entry Range 0-5000	2.4E-06	2.36E-06	2.552E-06	2.3652E-06	2.43932E-06	2.437004E-06
Naive LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	0.000119088	0.0001312476	0.00013108968	0.00013871316
Naive LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	0.000164896	0.0001803724	0.000182529	0.00019188204
Naive LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	0.00017714	0.0001945204	0.00019784904	0.00019357844

From the table, excluding the trivially fast 500 access trials, we can see that the average running time of a single cache access for the LRU varies between roughly 5.4e-7 and 1.0e-6 seconds; that's a difference of about half a microsecond. LRU access times are worse on larger entry ranges, but that coincides with a higher expected cache miss rate, and so is not unexpected. These values hold for both cache sizes. However, the same cannot be said for the Naive LRU; for the smaller cache, across all cache entry ranges (again excluding the trivially fast 500 access trials), the average access time is between 1.5 to 2 microseconds, while the average access time for the larger cache is between 1.1. and 1.9 tenths of a millisecond. That difference is once again two orders of magnitude, which falls in line with a linear growth for execution time.

CONCLUSION

From the above, it appears that the LRU implementation is much closer to constant runtime than the Naive LRU implementation, so much so that I would argue it appropriate to list as O(1). This confirms the claim, which because I can no longer seem to find it may as well be apocryphal, that the addition of a hash map for cache access reduces the asymptotic runtime of a naive LRU implementation from linear to constant time.

FUTURE WORK

It is worth noting that the LRU cache implementation used in these experiments in theory must use at least twice as much memory as the Naive LRU; the unordered hash map that is improving its access times must contain an equal number of entries as there are nodes in the cache list, so for larger caches my LRU implementation may be prohibitively large for certain applications. The Pseudo LRU (PLRU) cache replacement policy was created to emulate the performance of an LRU policy but with less memory overhead, so I intend to implement and run my LRU implementation against a PLRU policy and see whether or not the PLRU is a worthwhile alternative.

Additionally, I should note that some of my assumptions for the tests above were wrong; namely, the projected cache hit rate was much higher on the last tests of the 10000 entry caches. I expected that with a cache entry range of 0-500000, a 10000 entry cache would have a similar hit rate as a 100 entry cache with a cache entry range of 0-5000; this assumption proved not entirely unreasonable, as for both cache sizes, the hit rate was about 80% for the 1.25x

cache entry range and about 40% for the 2.5x cache entry range. Instead, the hit rate was roughly 15 times better for the 10000 entry cache, at roughly 30% hits vs 2% hits for the smaller cache. These results hold true for both cache replacement policies. A table with the full average hit rates for my tests can be seen below.

Average Cache Hit Ratio

	500 Accesses	5000 Accesses	50000 Accesses	500000 Accesses	5000000 Accesses	50000000 Accesses
LRU, Cache Size 100, Entry Range 0-125	0.7024	0.7806	0.79944	0.7997724	0.80010648	0.800029924
LRU, Cache Size 100, Entry Range 0-250	0.3492	0.39464	0.39918	0.3998628	0.39999236	0.400064592
LRU, Cache Size 100, Entry Range 0-5000	0.0164	0.01988	0.019844	0.0201952	0.02011948	0.020115892
LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	0.691168	0.7998564	0.81073352	0.81179702
LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	0.384352	0.4307684	0.43554824	0.436077416
LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	0.269884	0.3019184	0.30464484	0.3050614
Naive LRU, Cache Size 100, Entry Range 0-125	0.6724	0.7896	0.79644	0.799736	0.79996436	0.79999992
Naive LRU, Cache Size 100, Entry Range 0-250	0.364	0.3924	0.400576	0.4001224	0.39989304	0.399997992
Naive LRU, Cache Size 100, Entry Range 0-5000	0.0252	0.02072	0.020372	0.020108	0.02009064	0.020122148
Naive LRU, Cache Size 10000, Entry Range 0-12500	n/a	n/a	0.6896	0.7993468	0.81060244	0.811759132
Naive LRU, Cache Size 10000, Entry Range 0-25000	n/a	n/a	0.384444	0.4308584	0.43558088	0.436073184
Naive LRU, Cache Size 10000, Entry Range 0-500000	n/a	n/a	0.26996	0.3016512	0.30472196	0.305101736

What this means is that for future tests, it may be worthwhile to increase the size of the “mostly misses” cache entry range on a larger cache to achieve similar hit rate performance between cache sizes.