LRU vs FIFO

Using lru_trace (./sim -f lru_trace -m 4 -s 15 -a lru/fifo)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	10	10	0	6	6	50%
FIFO	6	14	3	7	10	30%

Using fifo_trace (./sim -f fifo_trace -m 4 -s 12 -a lru/fifo)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	6	9	0	5	5	40%
FIFO	6	9	0	5	5	40%

LRU advantages over FIFO:

- Repetitive page patterns are more efficient on LRU as they are prioritized in the cache. This is the case when these patterns occur at intervals smaller than that of the cache size.
- LRU is generally better as it favours repeating page patterns which is very common in many applications used today.
- LRU does not suffer from Belady's Anomaly, it actually improves when there's more cache memory available.

FIFO advantages over LRU:

- In scenarios where LRU and FIFO have similar hit and miss rates, FIFO is more cost efficient and is quicker in runtime as there's no processing needed. A typical scenario where this could occur is when repeating patterns are happening at intervals larger than the cache size making LRU perform the same as FIFO.

LRU vs CLOCK

Using lru_trace_2 (./sim -f lru_trace_2 -m 4 -s 15 -a lru/clock) (belady's anomaly)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	4	10	0	4	4	33.33%
CLOCK	2	12	1	5	6	16.67%

Using clock_trace (./sim -f clock_trace -m 4 -s 12 -a lru/clock)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	6	9	0	5	5	40%

CLOCK	6	9	0	5	5	40%

LRU advantages over CLOCK:

- Generally, randomly ordered repeating page patterns perform better on LRU as the order in which the clock algorithm is caching could in a way hurt the hit rate.
- CLOCK is at the end of the day an approximation of LRU which explains why the hit rates are very similar.
- However, LRU does not suffer from Belady's Anomaly while CLOCK does which makes it have edge case patterns which would cause many misses/faults.

CLOCK advantages over LRU:

- Some specific page patterns could perform much similarly on CLOCK and LRU as a pointer rotates around in the same order that they were added to the cache to figure out which page to evict/discard. This means that some patterns will result in similar hit rates on CLOCK.
- In the scenario where CLOCK and LRU are performing similarly in terms of hit rate, the CLOCK page replacement algorithm is more cost efficient as it only requires a pointer so its space complexity is O(1) while LRU's space complexity is O(M) where M is the size of the cache memory. LRU is O(M) in space because it has to keep track of when each page was last accessed. Over the long run, CLOCK could save more memory and money.

LRU vs ARC

Using lru_trace (./sim -f lru_trace -m 4 -s 15 -a lru/arc)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	10	10	0	6	6	50%
ARC	10	10	1	5	6	50%

Using arc_trace (./sim -f arc_trace -m 5 -s 15 -a lru/arc)

Algorithm	Hit Count	Miss Count	Clean Evictions	Dirty Evictions	Total Evictions	Hit Rate
LRU	4	19	3	11	15	17.39%
ARC	6	17	4	8	12	26.09%

LRU advantages over ARC:

- At times when LRU is similar in performance in terms of hit rates, it is actually quicker and saves more money as it does not take as much processing time to figure out whether or not a hit or miss happened. Arc has to check T1, T2, B1, B2 to cover different edge cases and scenarios. Over time, LRU could potentially save more money.

ARC advantages over LRU:

- ARC dynamically chooses between LRU and LFU which is a good thing because sometimes the intervals in which repeating patterns emerge are larger than the size of the cache so having the ability to dynamically shift to another algorithm makes ARC a better option for more generic use cases.
- Shifting between LRU and LFU allows ARC to better handle different patterns over the course of different applications and use cases.