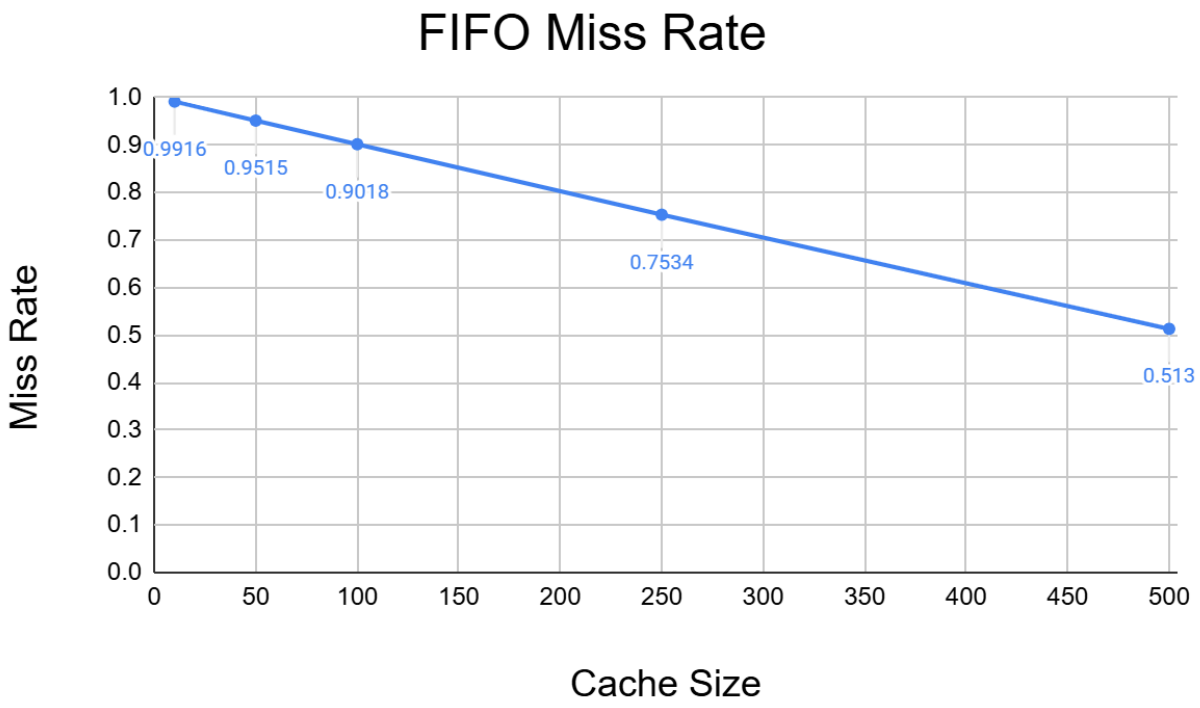
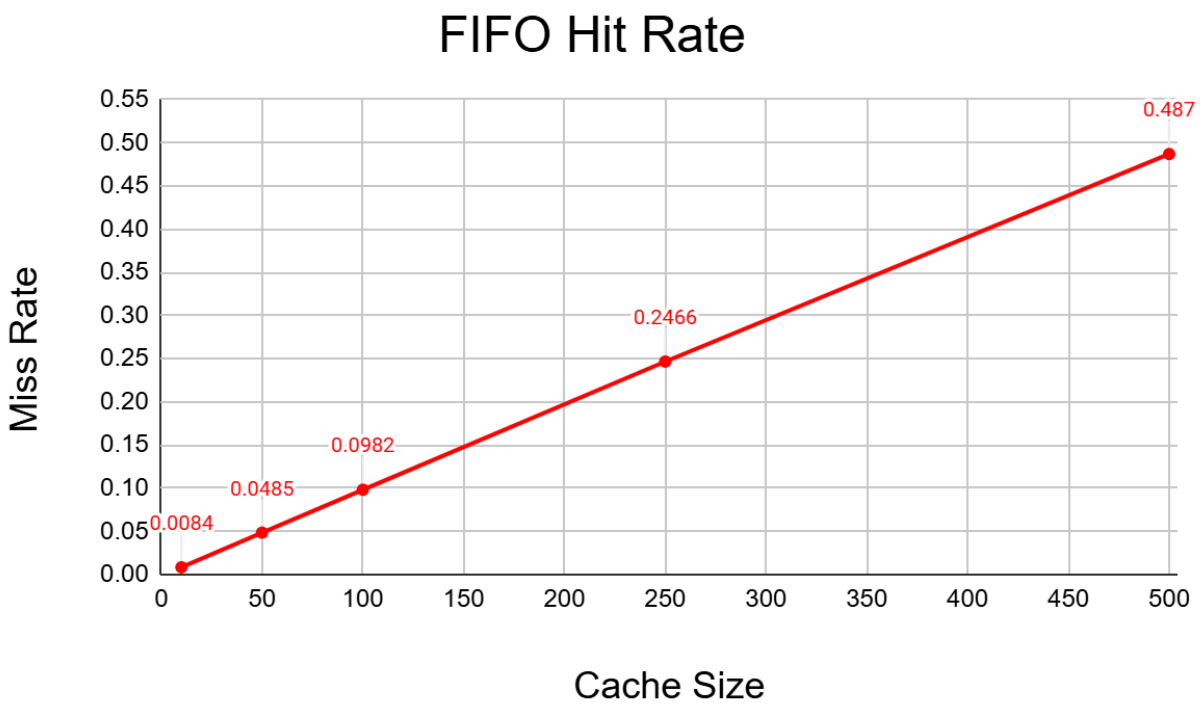
FIFO Algorithm:



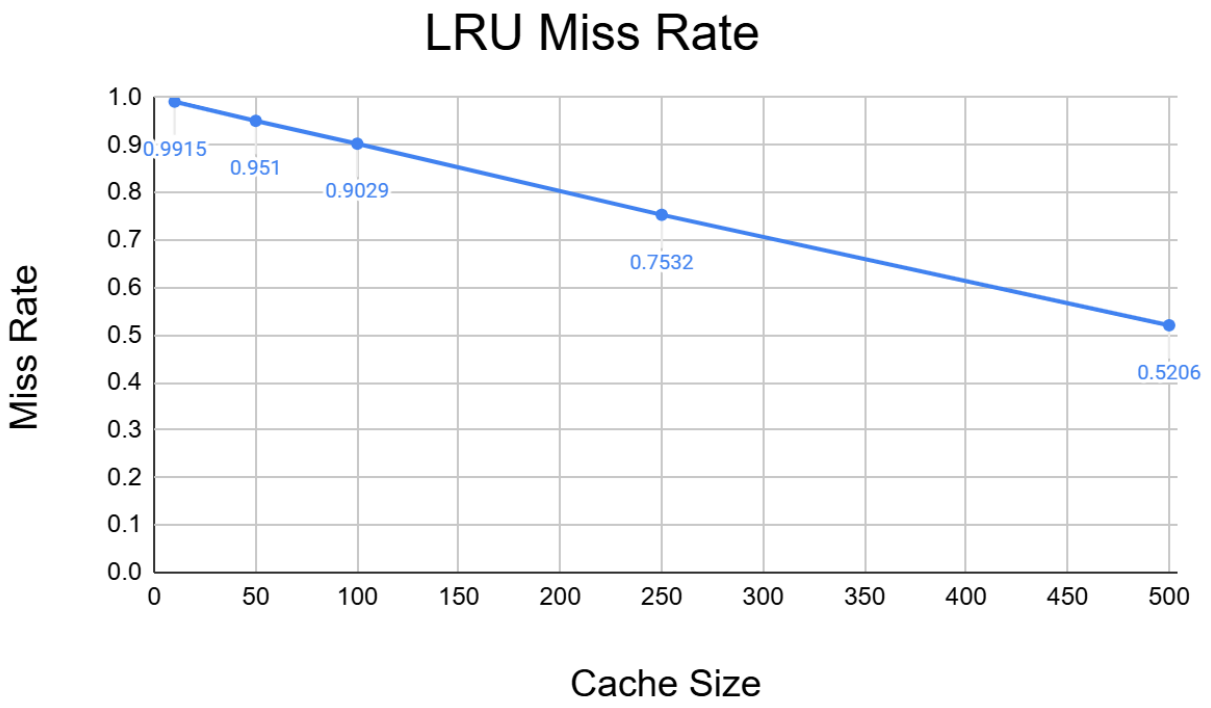


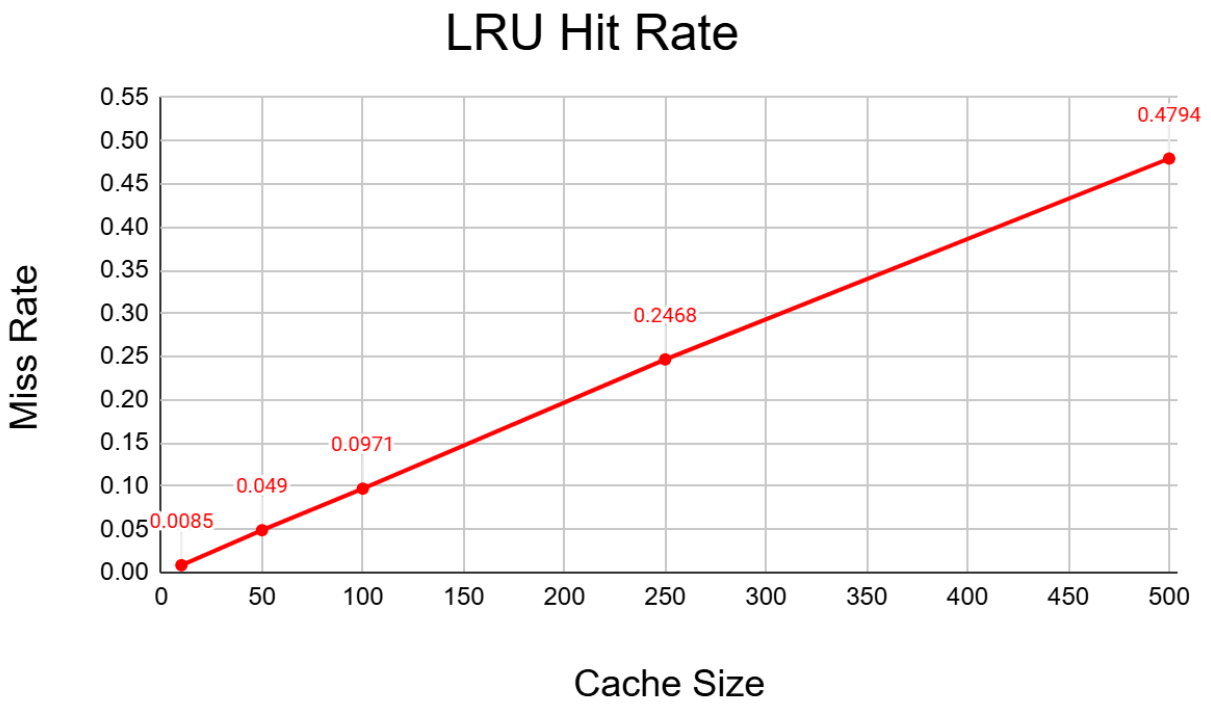
FIFO Algorithm Explanation/Observations:

The FIFO page replacement algorithm works by selecting the oldest page added to memory to be the victim page to be replaced. When a page request is made, the algorithm first searches for the page in the cache (an array of page structs with corresponding page numbers in memory). An is used to keep track of the oldest page that was added to the cache, so the cache behaves as a FIFO queue, where the most recently added pages are further to the right of the oldest page index. If a page isn't found, then the page at the oldest page index in the cache array is selected as the victim page, and its page number is replaced with the page number of the newly requested page. The oldest page index is then increased by 1, as the next oldest page will always be the page immediately to the right of the oldest page (the youngest will be immediately to the left). The oldest page index is initially set to 0, so if they array is empty or not completely filled, all of the unfilled slots will eventually filled from left to right based on how the FIFO algorithm works.

The reason that the FIFO miss rate decreases and hit rate increases as the cache size increases is likely because a larger cache size means that more pages can fit in the cache array. This means that the FIFO algorithm will have more cache hits (finds the page in the cache) and less cache misses (doesn't find the page in the cache) since a larger cache means it's more likely that the page the FIFO algorithm is looking for is already inside the cache than for smaller-sized caches.

LRU Algorithm:



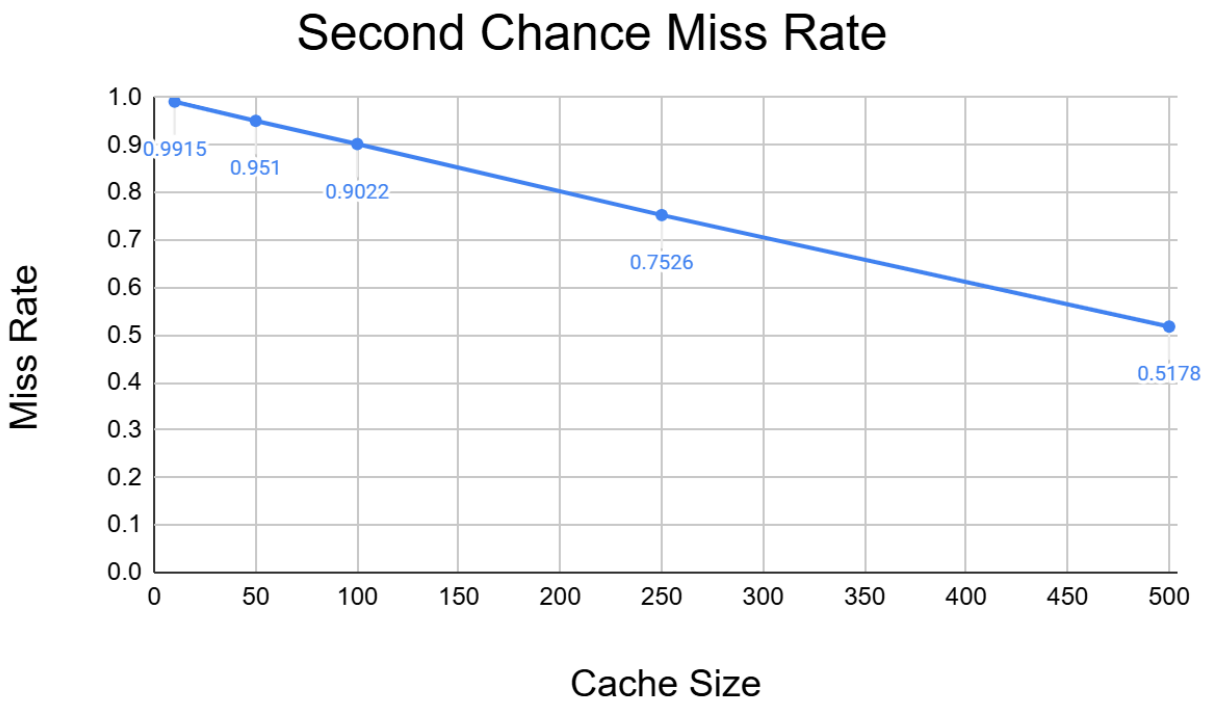


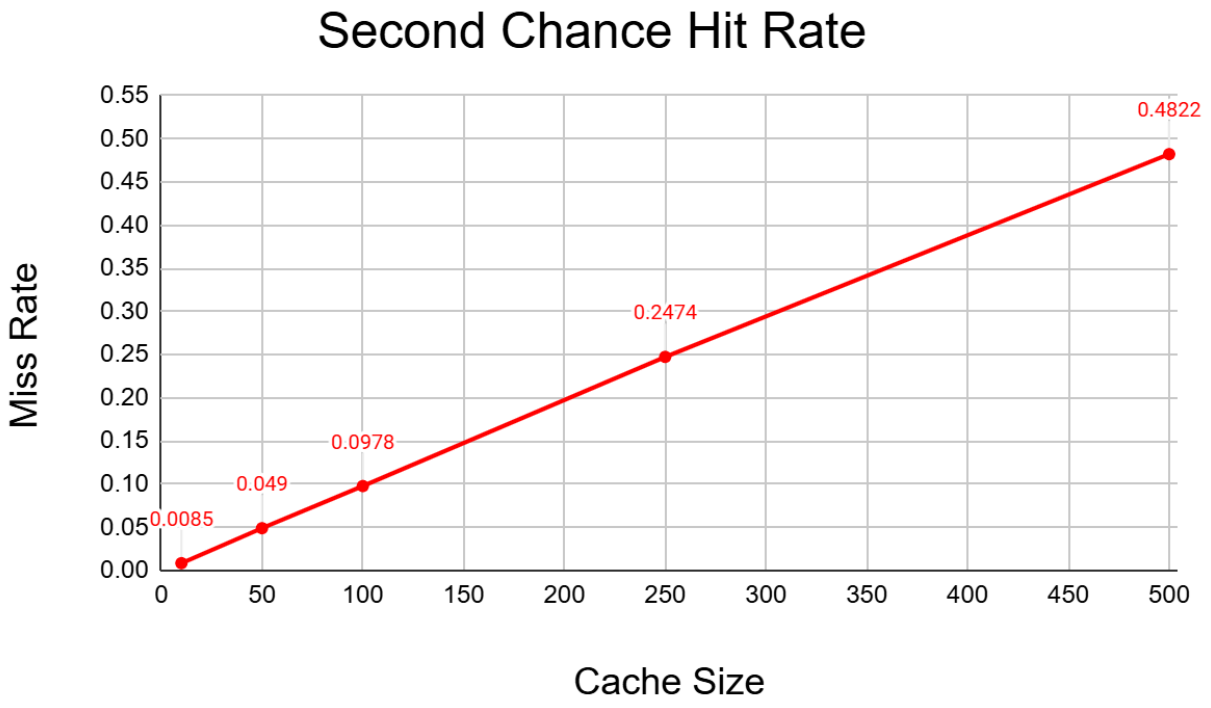
LRU Algorithm Explanation/Observations:

The LRU page replacement algorithm works by selecting the least recently accessed page as the victim page to be replaced. When a page request is made, the algorithm first searches for the page in the cache. The algorithm uses a counter to record the total number of requests that have currently been made. Each page has its own page number and lastRequested variable, which stores the number of the last request that the page was accessed on (ex: was last requested on request 500, and current number of requests made is 1000). If the page isn't found in the cache, the algorithm finds the least recently accessed page by looking for the page with the lowest lastRequested value (it does this simultaneously while it searches for the page in memory) and then replaces the page number of that page with the page number of the newly requested page, and updates its lastRequested variable to the current number of requests made. For empty spaces in the cache, there's still a page struct in them, but they have their page numbers set to -1 and their lastRequested variables set to 0, so while there are still unfilled slots the LRU algorithm will fill these slots from left to right.

The LRU miss rate decreases and hit rate increases as the cache size increases probably because a larger cache size means that more pages can fit in the cache array. This means that the LRU algorithm will have more cache hits and less cache misses since a larger cache means it's more likely that the page the LRU algorithm is looking for is already inside the cache than for smaller-sized caches.

Second Chance Algorithm:





Second Chance Algorithm Explanation/Observations:

The second chance algorithm works similarly to the FIFO algorithm, except instead of immediately removing the oldest page, it'll only remove a page if it has been selected for removal 2 times in a row without being accessed by a memory request (so it's like each page is given a second chance). An index somewhat similar to the index used for a FIFO queue will be used for selecting the victim, although instead of selecting the oldest page it'll just select a victim on the second time in a row that the algorithm selects a page. Each page struct has a pageHit variable, which is set to 1 when a page is accessed and set to 0 if a page is selected for removal. If a page is selected for removal and its pageHit value is 1, then the value is just decreased down to 0 and the algorithm looks for another victim page by increasing its search index. If a page is selected and the pageHit value is 0, then the page is the victim page and its page number is swapped with the number of the newly requested page. For empty spaces in the cache, there's still a page struct in them, but they have their page numbers set to -1 and their pageHit values set to 0, so while there are still unfilled slots the second chance algorithm will fill these slots from left to right.

The Second Chance miss rate decreases and hit rate increases as the cache size increases probably because a larger cache size means that more pages can fit in the cache array. This means that the Second Chance algorithm will have more cache hits and less cache misses since a larger cache means it's more likely that the page the Second Chance algorithm is looking for is already inside the cache than for smaller-sized caches.

Results Table:

	FIFO			LRU			Second Chance		
Cache Size	NumFaults	Miss Rate	Hit Rate	NumFaults	Miss Rate	Hit Rate	NumFaults	Miss Rate	Hit Rate
10	9916	0.9916	0.0084	9915	0.9915	0.0085	9915	0.9915	0.0085
50	9515	0.9515	0.0485	9510	0.951	0.049	9510	0.951	0.049
100	9018	0.9018	0.0982	9029	0.9029	0.0971	9022	0.9022	0.0978
250	7534	0.7534	0.2466	7532	0.7532	0.2468	7526	0.7526	0.2474
500	5130	0.5130	0.4870	5206	0.5206	0.4794	5178	0.5178	0.4822
Num Reqs:									
10000									