Ryan Chappelle

Professor Name

Class name

Date

Title

1. Beginning with the implementation of the First In First Out page replacement algorithm, not much needed to be changed from the provided code. We can see that the while loop runs if input is found from stdin. We then loop through our array of caches looking for that page number. If it is found, we break. Otherwise, we mark this as a page fault, and set the page that have been in the cache the longest as this new page number. We do this by keeping track of the oldest index, placeInArray. We reset this to (placeInArray + 1) % cacheSize so that it will keep looping. We added 1 so that it continues to iterate.
   1. Miss Rates: (10: 98.8%) (50: 94.8%) (100: 89.53%) (250: 75.19%) (500: 52.2%)
2. The implementation of Least Recently Used differs in the use of the time attribute of our ref\_page struct. This will be used to keep track of when a cache was most recently accessed. The next difference from the implementation of FIFO is within the for loop. If a page number is found in the cache, we set that cache’s time to index, which just iterates at the end of every while loop iteration. If the page number isn’t found, we loop through to find the cache with the lowest time (meaning it has been there the longest) and we replace it.
   1. Miss Rates: (10: 99.12%) (50: 95.48%) (100: 90.49%) (250: 75.48%) (500: 52.00%)
3. The implementation of Second Chance has changes in the same places as LRU. We first add a “flag” attribute to ref\_page. Then, if we find a cache with the correct page\_num, we set this flag to 1, meaning it has been accessed recently and gets a second chance next time it is looked at. If the page\_num is not found, we loop through the cache array until we find a cache.flag == 0. However, on the way we set every flag that is equal to 1 to 0. Once we find a cache.flag == 0 we replace it with the current page\_num.
   1. Miss Rates: (10: 98.82%) (50: 94.83%) (100: 89.65%) (250: 75.15%) (500: 52.41%)