GTU Department of Computer Engineering
CSE 312 - Spring 2022
Homework #2 Report

Hasan Mutlu
1801042673

IMPLEMENTATION

In the homework, I have simulated the page replacement algorithms. I have written seperate classes for each replacement algorithm and these classes were the abstarction of virtual memory. Inside these classes, the are some data fields that abstracts the memory components. These are:

- ➔ numArray[ARRSIZE]: This array contains the numbers that are supposed to be sorted. It can be thought of like the disk.
- ➔ pages[PAGES]: This array contains the starting index addresses of pages that are currently in the memory. The block size of each page is determined by macro INTPERPAGE. This value must be divisor of ARRSIZE value. The INTPERPAGE*PAGES value gives the size of the memory.

The numbers in the array are requested and manipulated by get and set methods. These methods checks the page table to see if the requested index is in the range of loaded pages in the page[] array. In order to get a hit, when the requested number is at the ith index of the numArray, there should bevalue j in pages[] array, which satisfies condition:

- ➔ i >= pages[j]*INTPERPAGE && i <= pages[j]*INTPERPAGE+INTPERPAGE-1

For example, suppose that we have any number of pages in the memory, and page block size is 10. (In other words, INTPERPAGE is 10) If the index of the number that is requested for sorting is 143, the value 14 must be in the pages[] array in order to get a hit. (143 >= 14*10 && 143 <= 14*10+10-1) Presence of address 14 in pages[] array means that the values from 140 to 149 are available in the memory.

If the required address isn't present in the pages[] array, the required page replacement algorithm is used to bring it to memory.

In both occasions, the counters hit and miss are incremented accordingly.

LRU

In LRU, there are couple of more data fields. These are refTime[PAGES] and clock. The clock is just a counter that is incremented in every memory request. The refTime[] array is parallel to the pages[] array and records the value of clock in ever memory access. In other words, if pages[i] is accessed, the value of the clock at that moment is recorded to refTime[i].

When a miss happens, the values in refTime are checked and the index of the lowest one, in other words the oldest one is kept. The page at this address in page[] array is replaced by the requested number's page address.

FIFO

In FIFO, there isn't any additional fields. When a page fault happens, the page[] array is shifted and the requested number's page block address is added in front of the array, like queue.

Second Chance

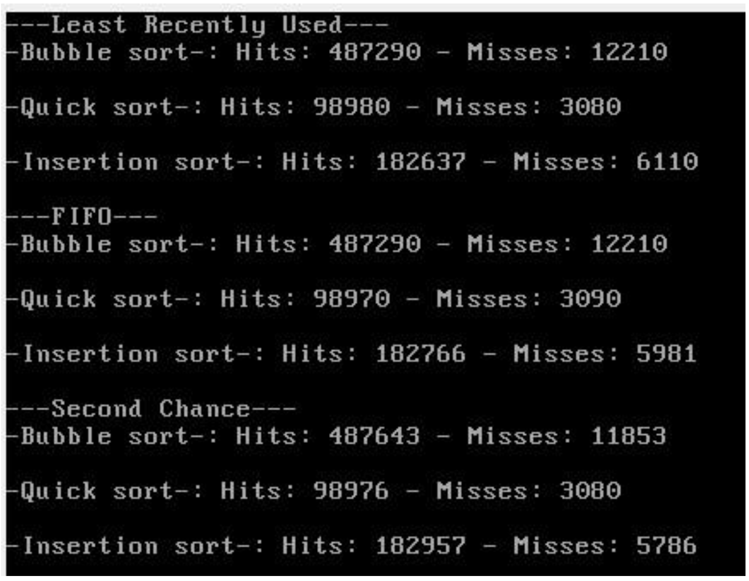
In second chance, I couldn't find a way to sign the reference bit when request comes and reset it when clock interrupt happens. Instead, I decided to use a list of recent references. There is refs[REFS] array. It is half of page table's size long. For example, if page table size is 10, then the most recent 5 references to pages are recorded. When miss happens, if an address at the end of the FIFO is marked as referenced in refs[] list, it is deleted from refs[] and given a second chance by being sent to the front of the list. At the end, the first page address that is at the end of the list, which isn't in the list of recent references, is replaced.

Definition of Test Scenario:

In test scenario, I the three sorting algorithms are operated on each page replacement algorithms. In the class constructor, the number array is created with not sorted values. A new class is created for each sorting algorithms and number of misses and hits are printed. I couldn't include time library so i couldn't calculate hit/miss rates per second. Also, number of pages loaded value would be equal to number of misses anyway. Furthermore, number of pages written back to the disk value is always equal to number of misses – page table size. So I excluded these properties in order to fit every output inside the host OS's small interface.

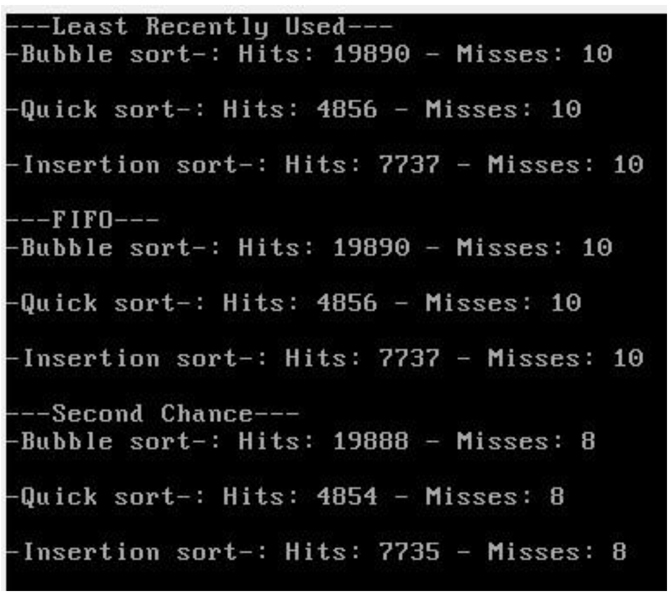
RUNNING AND RESULT

Array Size: 500, Page Table Size: 10, Page Block Size: 10 (The disk is 5 times bigger than the memory.)



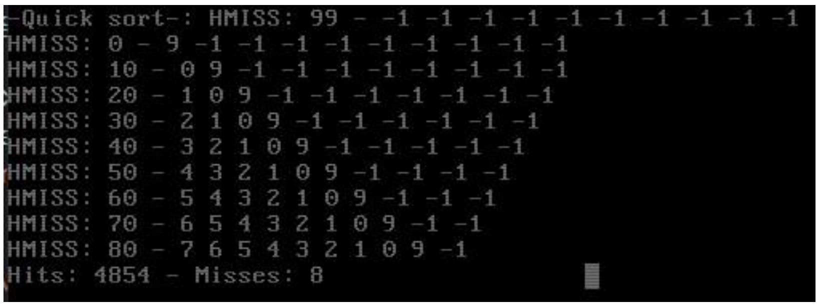
```
---Least Recently Used---
-Bubble sort-: Hits: 487290 - Misses: 12210

-Quick sort-: Hits: 98980 - Misses: 3080

-Insertion sort-: Hits: 182637 - Misses: 6110

---FIFO---
-Bubble sort-: Hits: 487290 - Misses: 12210

-Quick sort-: Hits: 98970 - Misses: 3090

-Insertion sort-: Hits: 182766 - Misses: 5981

---Second Chance---
-Bubble sort-: Hits: 487643 - Misses: 11853

-Quick sort-: Hits: 98976 - Misses: 3080

-Insertion sort-: Hits: 182957 - Misses: 5786
```

Array Size: 100, Page Table Size: 10, Page Block Size: 10 (The disk size is equal to memory.)



```
---Least Recently Used---
-Bubble sort-: Hits: 19890 - Misses: 10

-Quick sort-: Hits: 4856 - Misses: 10

-Insertion sort-: Hits: 7737 - Misses: 10

---FIFO---
-Bubble sort-: Hits: 19890 - Misses: 10

-Quick sort-: Hits: 4856 - Misses: 10

-Insertion sort-: Hits: 7737 - Misses: 10

---Second Chance---
-Bubble sort-: Hits: 19888 - Misses: 8

-Quick sort-: Hits: 4854 - Misses: 8

-Insertion sort-: Hits: 7735 - Misses: 8
```

There is a slight bug with second chance algorithm. It givesthe result as 8, which should’ve been 10. It was a weird bug, I couldn’t find the source of it. In the debugging screen, it can be seen that it has missed 10 times.



```
-Quick sort-: HMISS: 99 - -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
HMISS: 0 - 9 -1 -1 -1 -1 -1 -1 -1 -1 -1
HMISS: 10 - 0 9 -1 -1 -1 -1 -1 -1 -1 -1
HMISS: 20 - 1 0 9 -1 -1 -1 -1 -1 -1 -1
HMISS: 30 - 2 1 0 9 -1 -1 -1 -1 -1 -1
HMISS: 40 - 3 2 1 0 9 -1 -1 -1 -1 -1
HMISS: 50 - 4 3 2 1 0 9 -1 -1 -1 -1
HMISS: 60 - 5 4 3 2 1 0 9 -1 -1 -1
HMISS: 70 - 6 5 4 3 2 1 0 9 -1 -1
HMISS: 80 - 7 6 5 4 3 2 1 0 9 -1
Hits: 4854 - Misses: 8
```