Group Name: JAME
Group Members: John Villaflor, Andrew Bub, Molly Pierce, Erica Boyd
Operating System Principles
Project 5: Virtual Memory

Due: Tuesday, April 17 for 5% EC

In your own words, briefly explain the purpose of the experiments and the experimental setup. Be sure to clearly state on which machine you ran the experiments, and exactly what your command line arguments were, so that we can reproduce your work in case of any confusion.

The purpose of the experiments is to figure out how to move memory around from disk to physical memory/frames and vice versa based upon a virtual memory mapping. Also, the goal is to implement different strategies on how to evict from physical memory (because space is limited), and to determine which strategy results in the "best" performance (page faults, disk reads, and disk writes).

Our program was set up so we were only editing the page_fault_handler function. The function first reads in a page entry, and checks the permissions of this entry. If there are no permission bits, and there is an empty frame open in physical memory, then the page is read into physical memory and permission bits are changed to read. The page's reference to the proper physical frame number is set and the number of disk reads increments by one. If there are no empty frames open in physical memory, the program goes into one of the three eviction methods - random, fifo, and custom. Random uses a random seeding to evict a certain page out of physical memory. Fifo (first in first out) relies on the implementation of a queue, where the page corresponding to front of the queue is evicted first (since it was added first). Our custom method is explained below. If the page that is evicted has a dirty bit, we write the data back to disk before evicting it and increase the number of disk writes. If the read permissions are set, and the page just does not have write permissions, the write permissions are given (which shows the the page is now dirty) and the program returns and an access fault is tallied.

We ran the experiments on the student00 and student04 machines. There are 4 command line arguments - the number of pages, number of frames, random/fifo/custom eviction strategy, and scan/sort/focus program execution. To run our program we ran the command: ./virtmem <*number of pages*> <*number of frames*> <*rand, fifo or custom*> <*scan, sort or focus*>.

Very carefully describe the custom page replacement algorithm that you have invented. Make sure to give enough detail that someone else could reproduce your algorithm, even without your code.

The custom page replacement algorithm that we implemented was known as NRU, or Not Recently Used. When a page fault is detected, the page number is added to an array of size nframes/10 (if the number of frames was less than 10 then we divide by 3). If a page fault is detected and the physical memory is full, we use rand() to determine which page number to evict from physical memory. If the selected page number is in our array, we choose a different

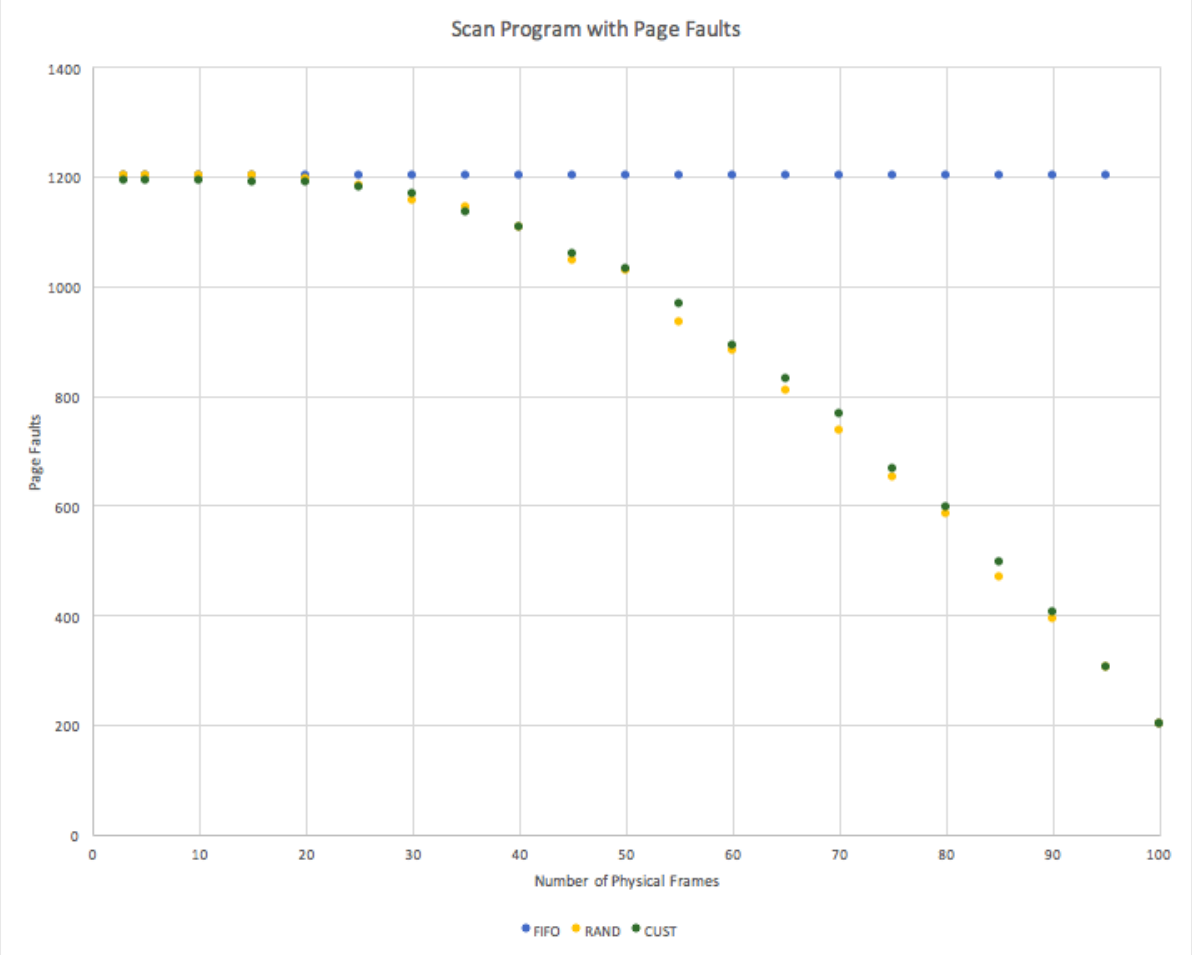
random number to evict - since it being in that array means it was recently added. This preserves the most recently accessed page numbers from being evicted.

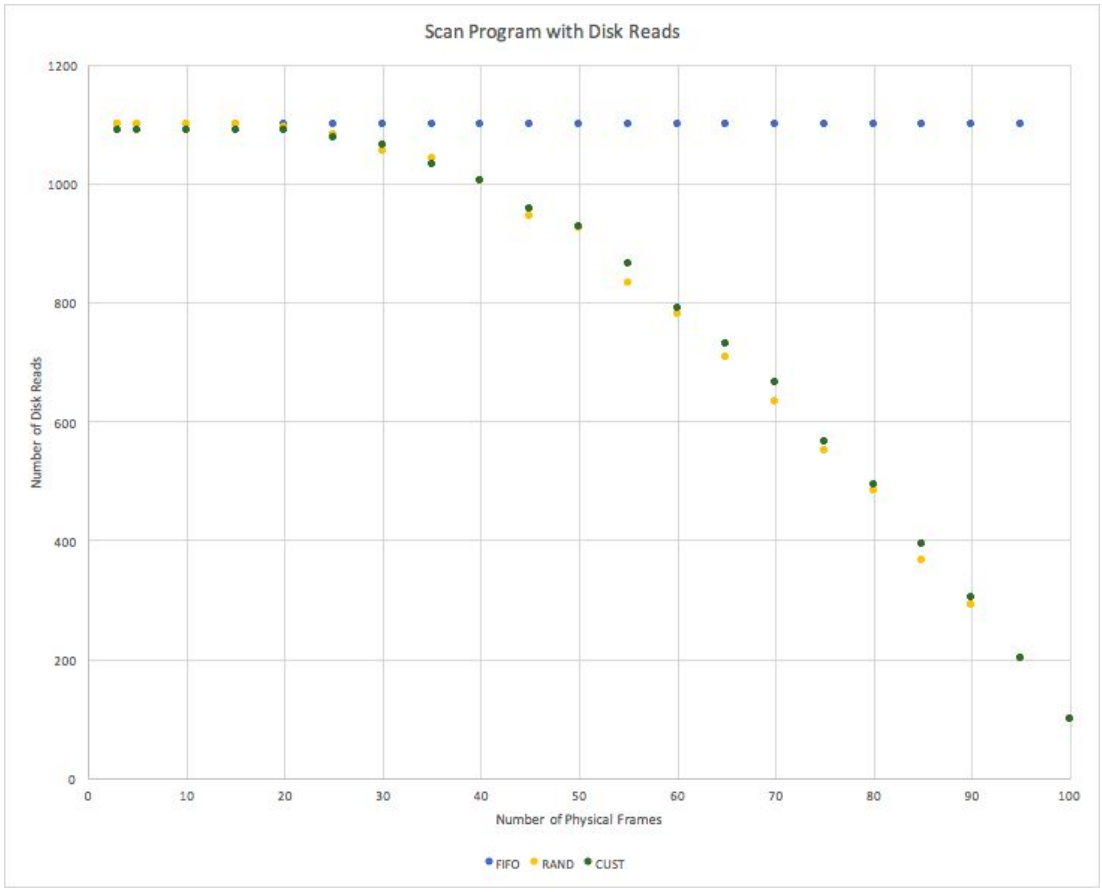
We decided to use this approach because sometimes, as we have discussed in class, pages that were recently used are used again in quick succession, so by preserving the most recently used page numbers from eviction, we can reduce the number of overall page faults.

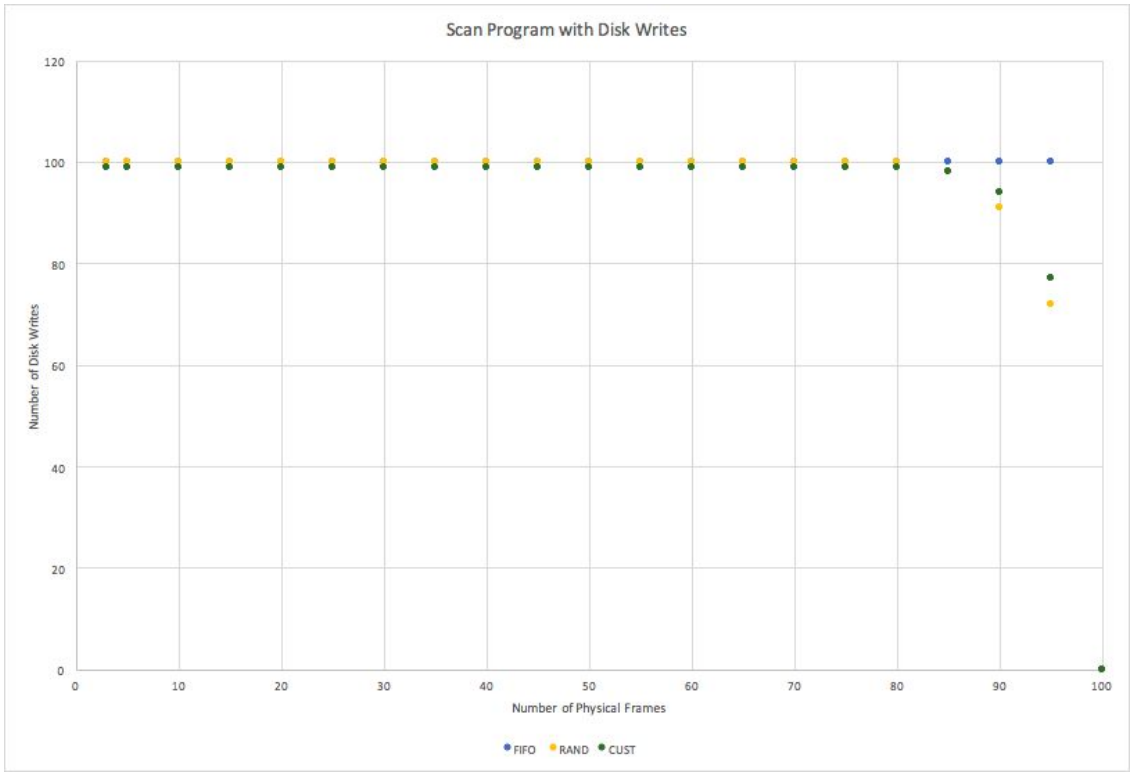
Measure and graph the number of page faults, disk reads, and disk writes for each program and each page replacement algorithm using 100 pages and a varying number of frames between 3 and 100. Spend some time to make sure that your graphs are nicely laid out, correctly labelled, and easy to read.

- In all cases, virtual pages = 100
- For all three methods, for all three programs, page faults = 200 at 100 physical frames
- *Note: Any graph "with Page Faults" means total page faults (real and access). Our program output distinguishes between real page faults and access faults; but, for the purpose of graphing and discussion, we summed the number of faults.*

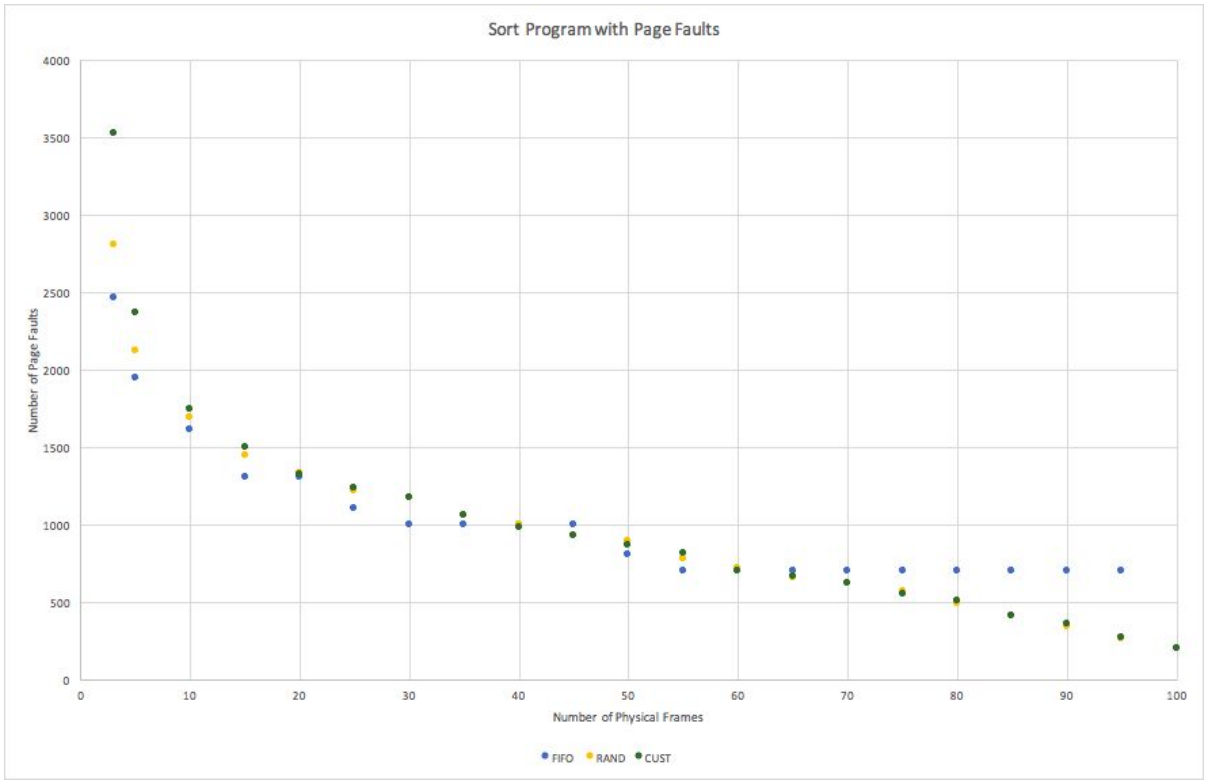
SCAN GRAPHS:

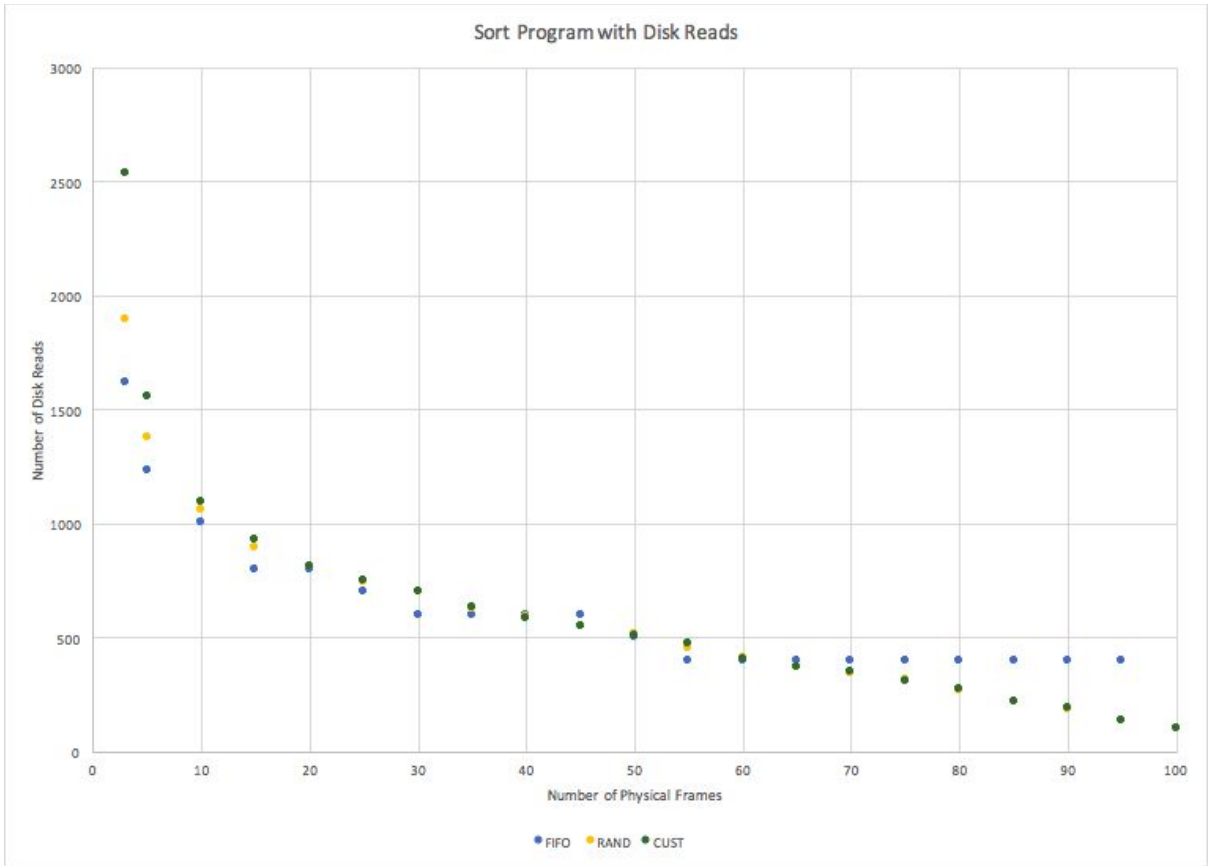


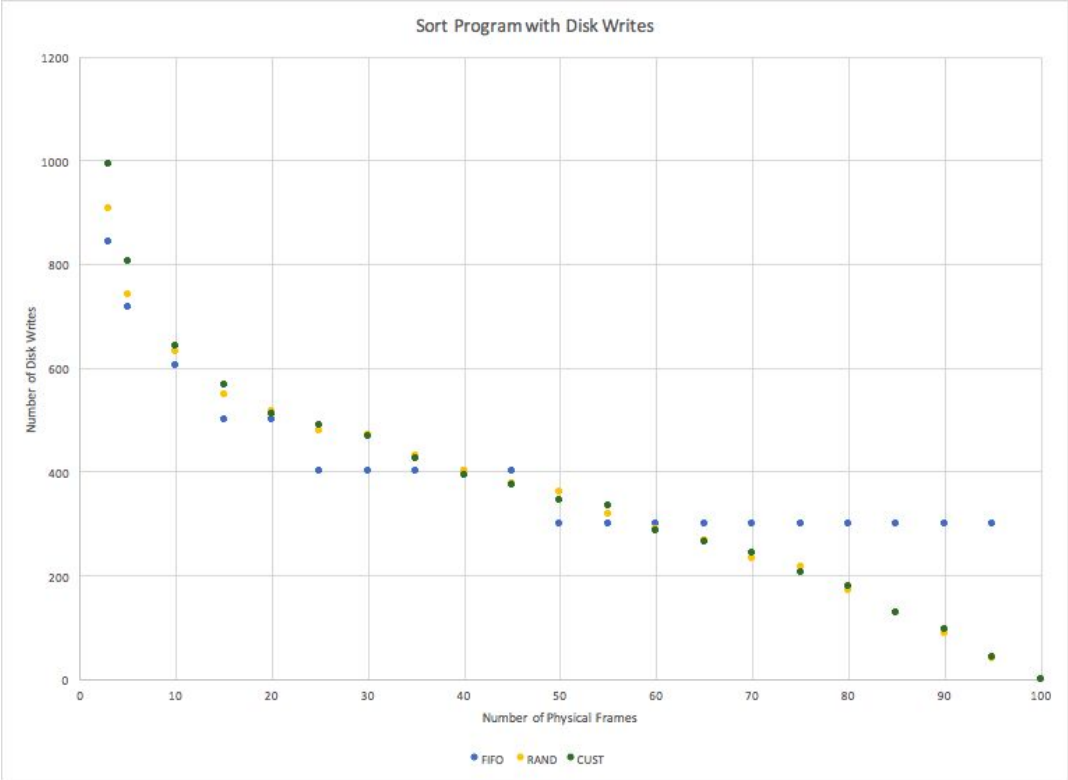

Scan Program with Disk Reads
Number of Disk Reads
Number of Physical Frames
FIFO
RAND
CUST


Scan Program with Disk Writes
Number of Disk Writes
Number of Physical Frames
FIFO
RAND
CUST

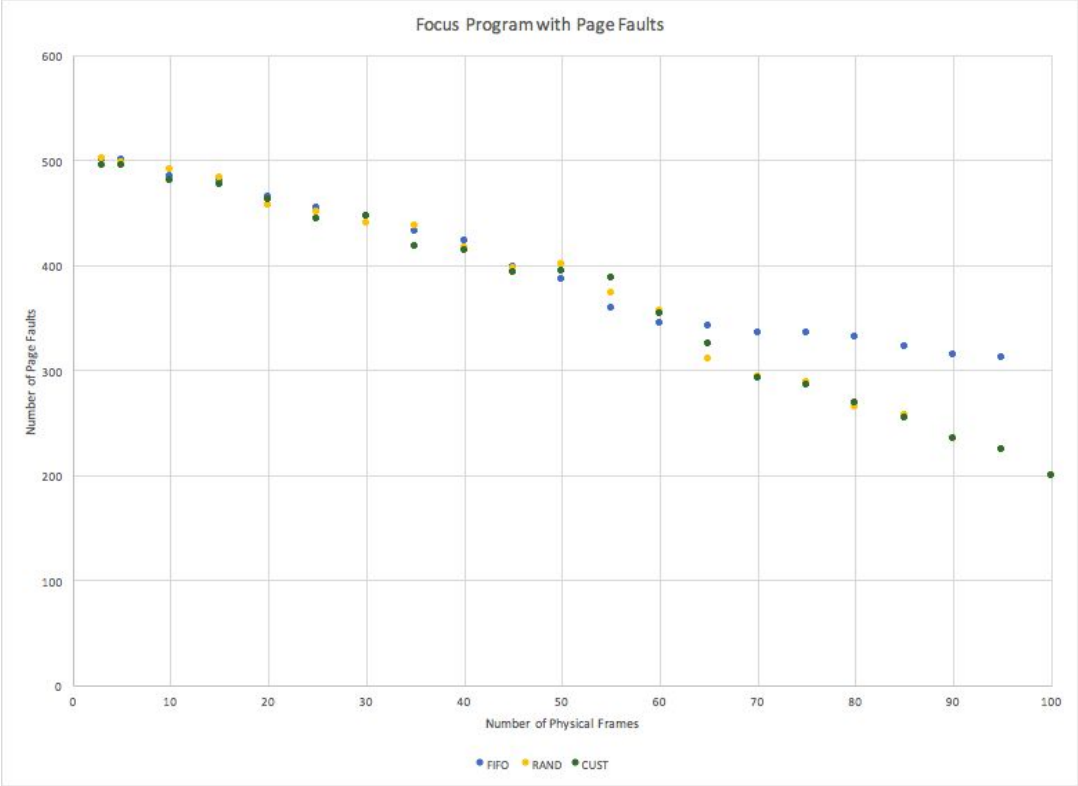
SORT GRAPHS:

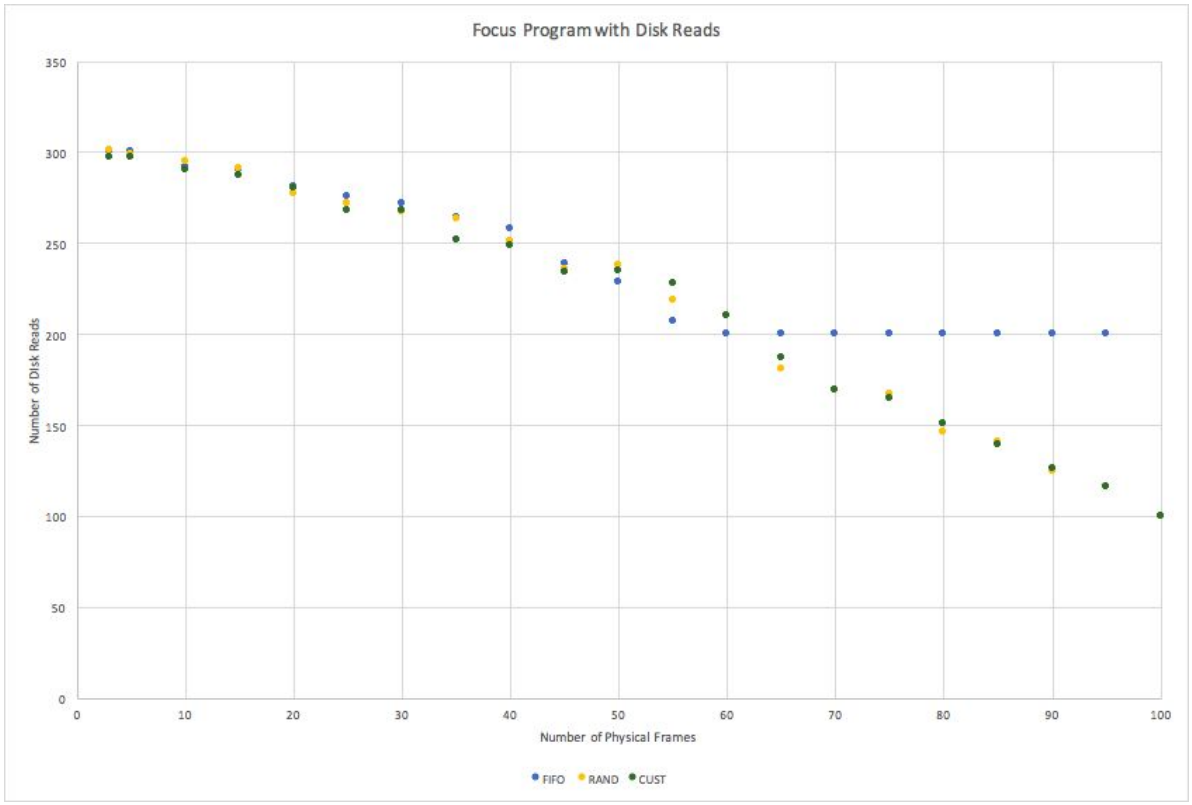

Sort Program with Page Faults
Number of Page Faults
Number of Physical Frames
FIFO
RAND
CUST

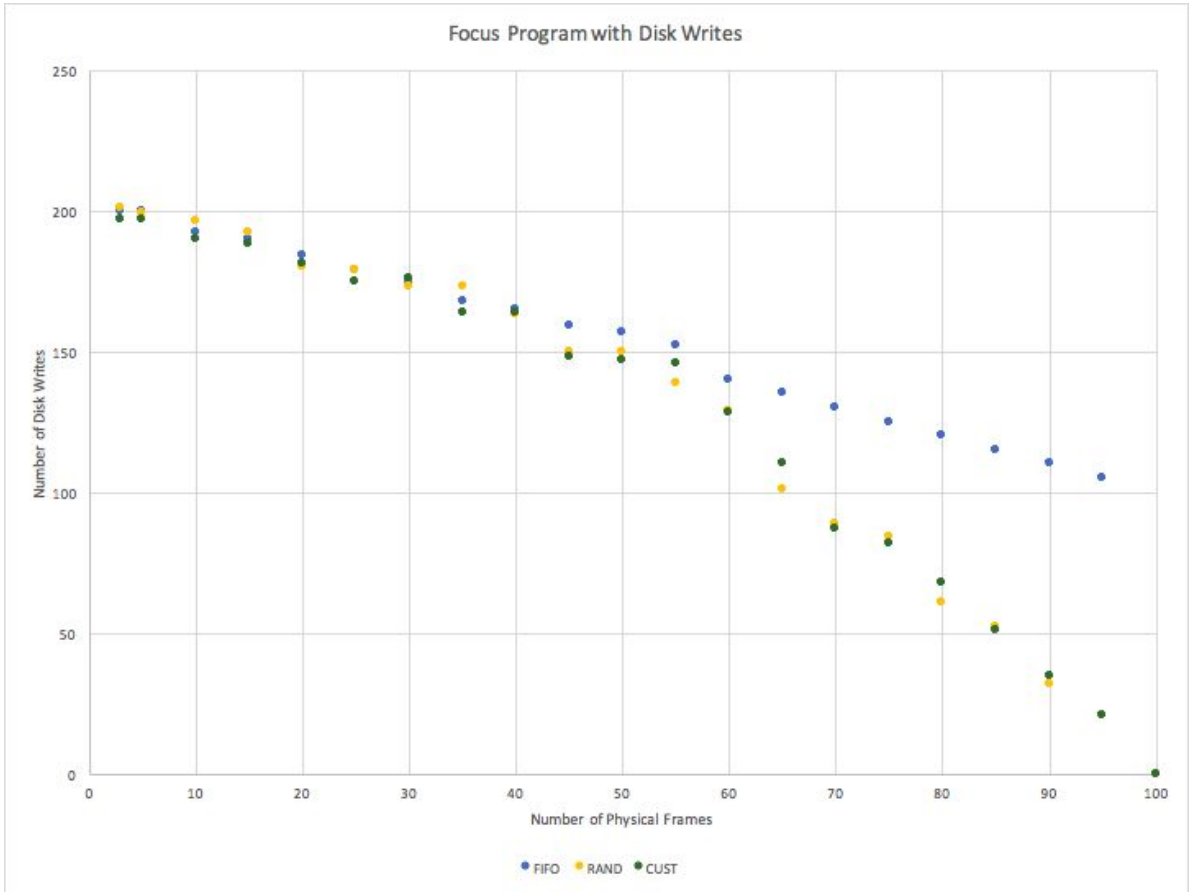

Sort Program with Disk Reads
Number of Disk Reads
Number of Physical Frames
FIFO
RAND
CUST



FOCUS GRAPHS:




Focus Program with Disk Reads
Number of Disk Reads
Number of Physical Frames
FIFO
RAND
CUST


Focus Program with Disk Writes
Number of Disk Writes
Number of Physical Frames
FIFO
RAND
CUST

Explain the nature of the results. If one algorithm performs better than another under certain conditions, then point that out, explain the conditions, and explain why it performs better.

SCAN:

As shown by our Scan graphs, the FIFO method seemed to perform the worst across the board, while rand and our custom method performed very similarly. From 0 up to 100 physical frames, the FIFO method produced an equal number of page faults, disk reads, and disk writes, no matter how many frames there were. Compared to our other methods, FIFO is much more structured and no matter how many physical frames there are, when scanning, the same actions will occur for whatever the application is trying to read or write. For page faults and disk reads, our rand and custom methods improve at a mostly linear rate as the number of physical frames increase. This would make sense because less pages need to be evicted as there is more room to hold them. For the disk writes, the number stays constant around 100 for rand and custom until the number of physical frames gets much closer to the number of virtual pages. This could occur because there will be much less evictions and therefore much less a chance that the page that is evicted will have a dirty bit.

SORT:

The Sort program graphs followed most of the other trends of the other graphs, including the decrease of page faults, disk reads and disk writes as the number of frames is increased. The difference was that the random eviction strategy tended to perform a lot better than both custom and FIFO across all three metrics. The FIFO strategy worked the best at lower physical frames, and then started to level out as the number of physical frames increased. The random and custom strategy started off with a significant more page faults, disk reads and disk writes at 3 and 5 physical frames, and then started to decrease at a consistent rate until the maximum number of physical frames were reached, which aligns with what we observed from scan and focus graphs.

FOCUS:

In general, across all three Focus graphs, as the number of frames increased, the number of page faults / disk writes / disk reads decreased. Also, as seen in the graphs, as the number of frames increased, the FIFO approach produced diminishing returns and levels out. However, random and custom continue to decrease in the number of page faults / disk writes /disk reads at a relatively linear rate. This is most likely because at high frame numbers, the FIFO Queue has the same behavior and therefore there are diminishing returns on performance. We see the best performance gains with disk reads and disk writes. Again, this would make sense because less pages need to be evicted as there is more room to hold them - and with less evictions there is a lesser chance of having to write back to disk.

GENERAL:

In general, the program performs better when there are more frames because it is more likely that there will be a hit, since there is more room, so less pages will need to be evicted. Also, FIFO performs the worst because of the manner in which it evicts linearly. Because these programs act in a linear manner, evicting something that was used the longest ago is counterintuitive because it is likely that item will actually be needed again in the near future. This is also why out custom method works so similarly to the rand method. The programs don't seem to access the same pages in clustered times, but rather iterate through linearly multiple times, making our buffer of recently used pages basically moot - but still much better than FIFO.