CSCI 5103 - OPERATING SYSTEMS
PROGRAMMING ASSIGNMENT-3
DESIGN DOCUMENT

GROUP MEMBERS:
Sai Pratyusha Attanti *(attan005@umn.edu, Student ID: 5656785)*
Navya Ganta *(ganta016@umn.edu, Student ID: 5673223)*

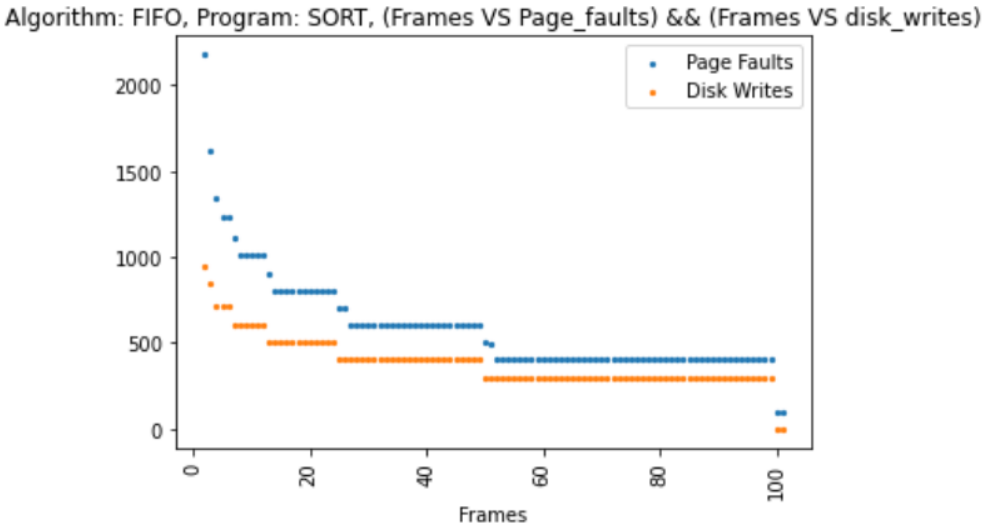
Instructions to compile and run the project:

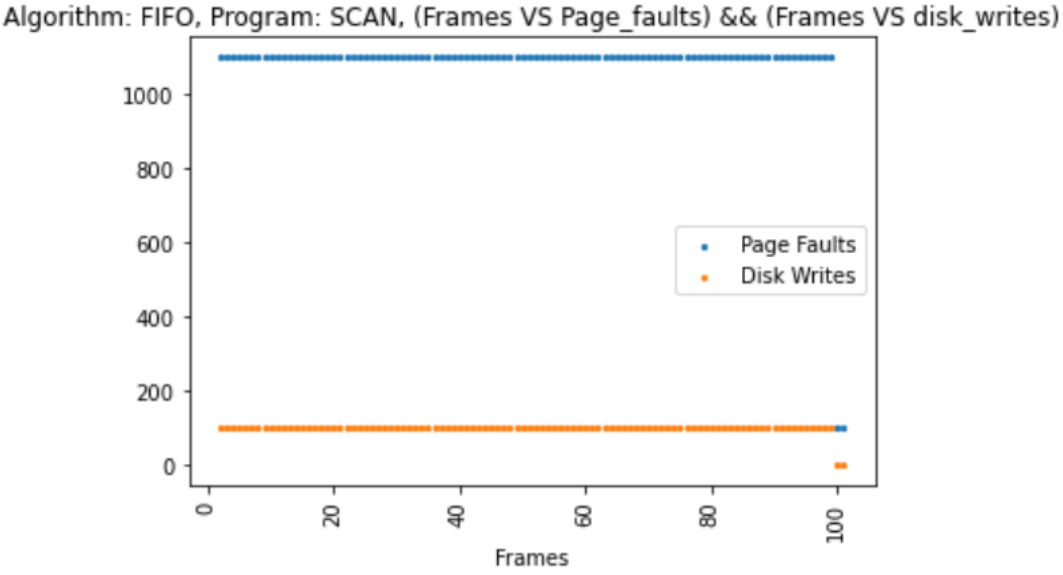
1. **To run individual tests:**
   - Run `make` to compile the code
   - To Run:
     `./virtmem <npages> <nframes> <rand|fifo|custom> <sort|scan|focus>`
2. **To Generate the data for the Graphs:**
   - Run `make` to compile the code
   - To Run:
     `./virtmem <rand|fifo|custom> <sort|scan|focus>`
   - The above command will run the tests for fixed pages(100) and varying frames(1-100) and generate the page faults and disk writes for each combination and store them in text files.
3. **To generate graphs from the above data:**
   - Run: `python3 graph.py`

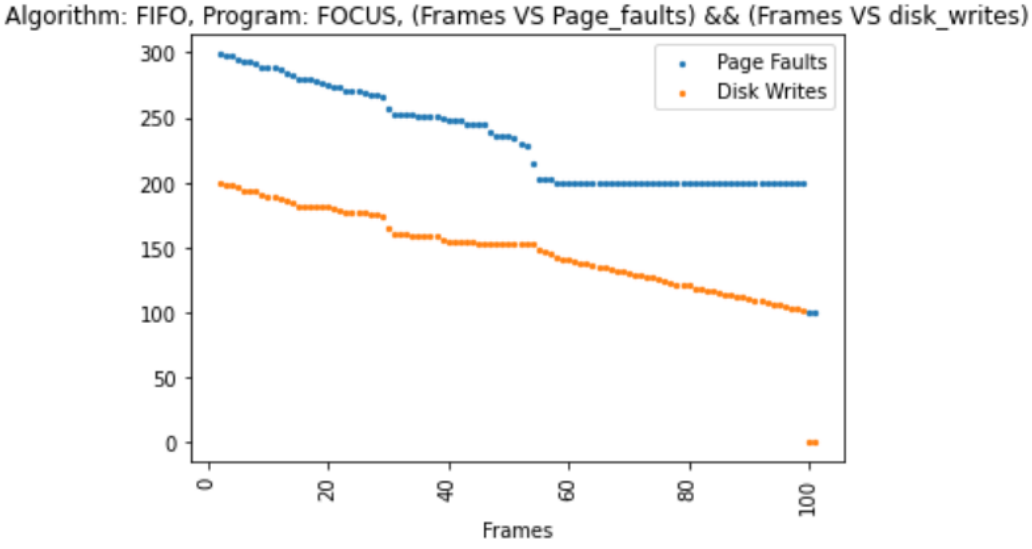
Custom page Replacement Algorithm:
In our custom page replacement algorithm, when a page fault occurs, from the set of pages in physical memory which has their write bit set we select one page randomly and replace it with the new page. And if there is no page with a write bit set the replacement algorithm then we select one of the pages randomly and replace it with the new page. The reason behind this idea is that the page with only the read bits set might be used for write operations in the future so it is better to have them in the main memory in case the write operations might take place on these pages, so that is why write bits are first considered for replacement when compared to the pages with just read bits set.

Purpose of the Experiments:
In this project, we are implementing three different page replacement algorithms i.e RANDOM, FIFO, and CUSTOM page replacement algorithms for different programs with different memory patterns.
For fixed pages(pages=100) and varying frames (frames = 1 to 100), we are comparing the performance of each replacement algorithm based on the number of page faults and disk writes to understand the behavior of the algorithm under varying frames.
Also for each program, we are comparing the performances of the page replacement algorithm to understand which algorithm to use based on their memory access patterns. All these experiments are done using the Vole2D virtual machine.

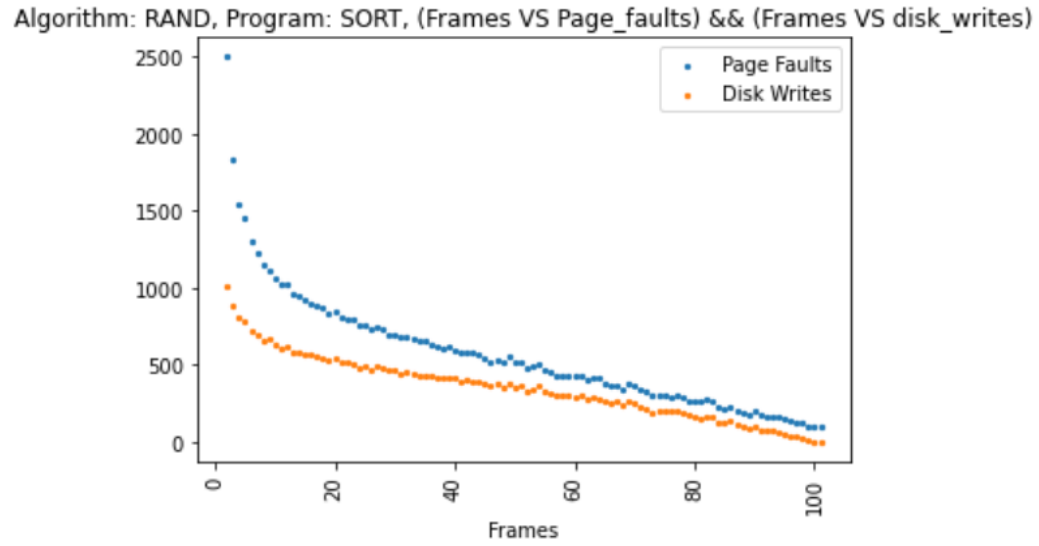
FIFO PAGE REPLACEMENT ALGORITHM

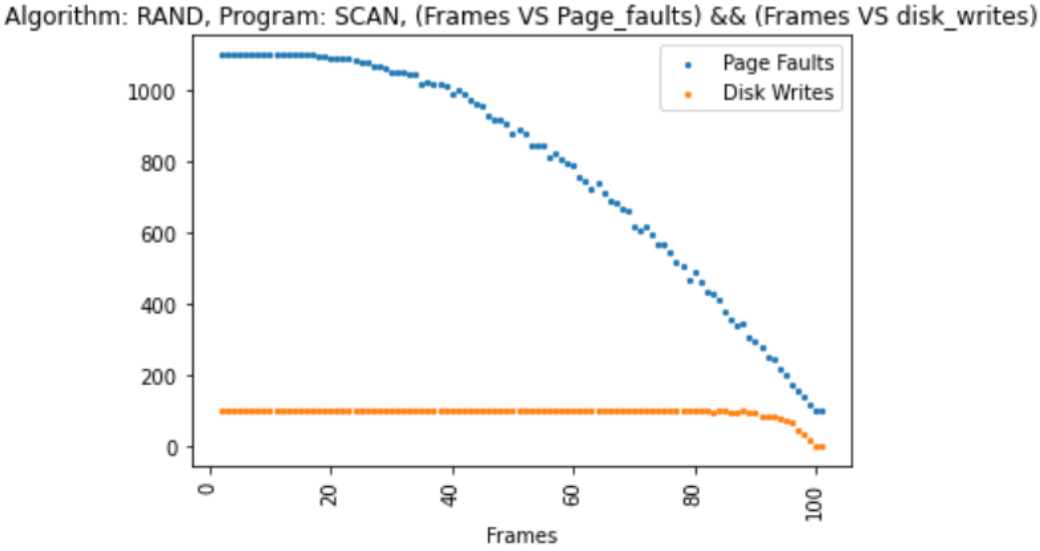


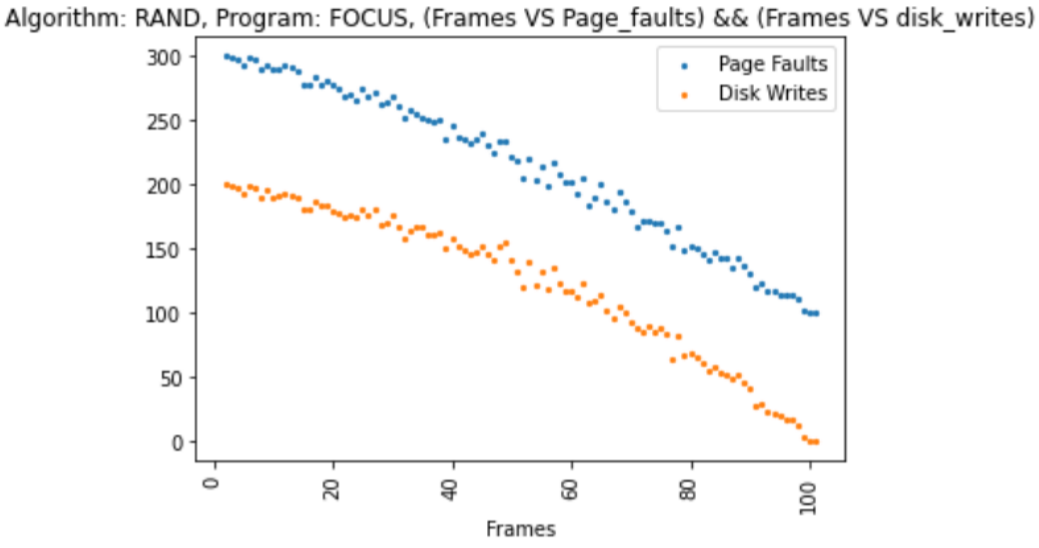




RANDOM PAGE REPLACEMENT ALGORITHM

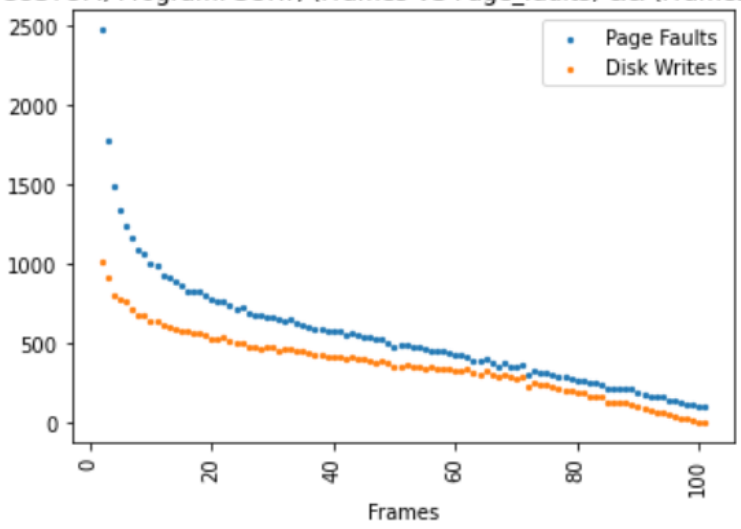




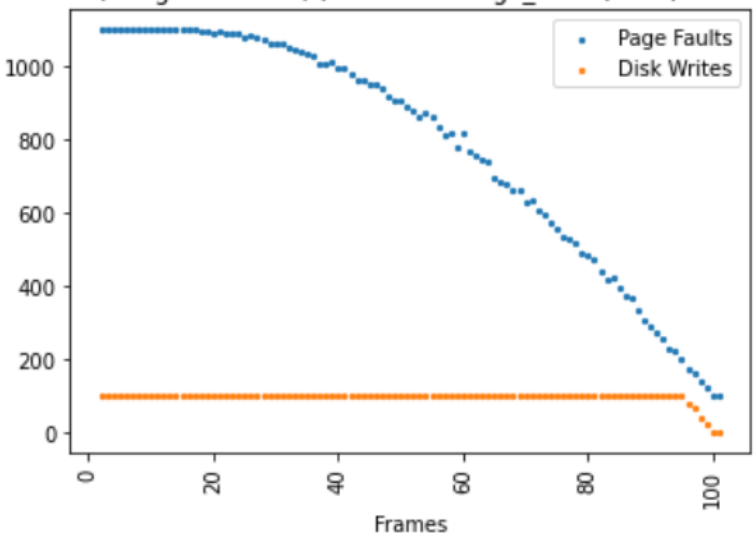


CUSTOM PAGE REPLACEMENT ALGORITHM

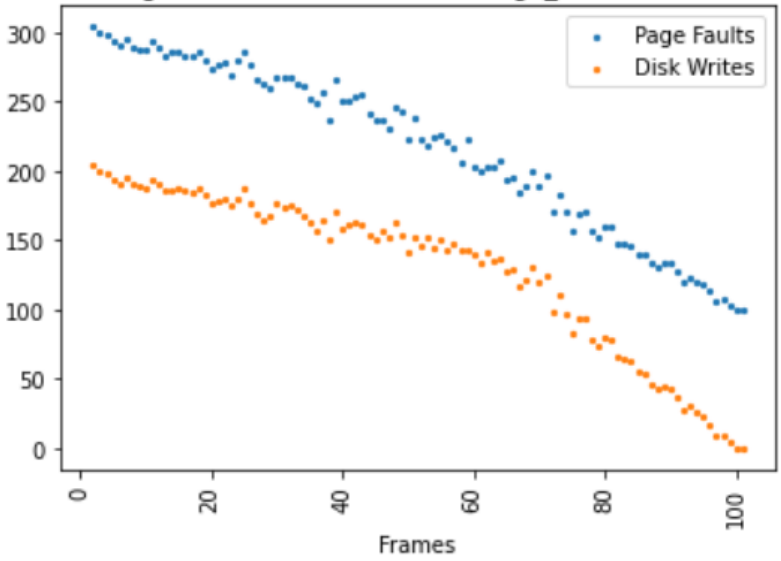
Algorithm: CUSTOM, Program: SORT, (Frames VS Page_faults) && (Frames VS disk_writes)



Algorithm: CUSTOM, Program: SCAN, (Frames VS Page_faults) && (Frames VS disk_writes)

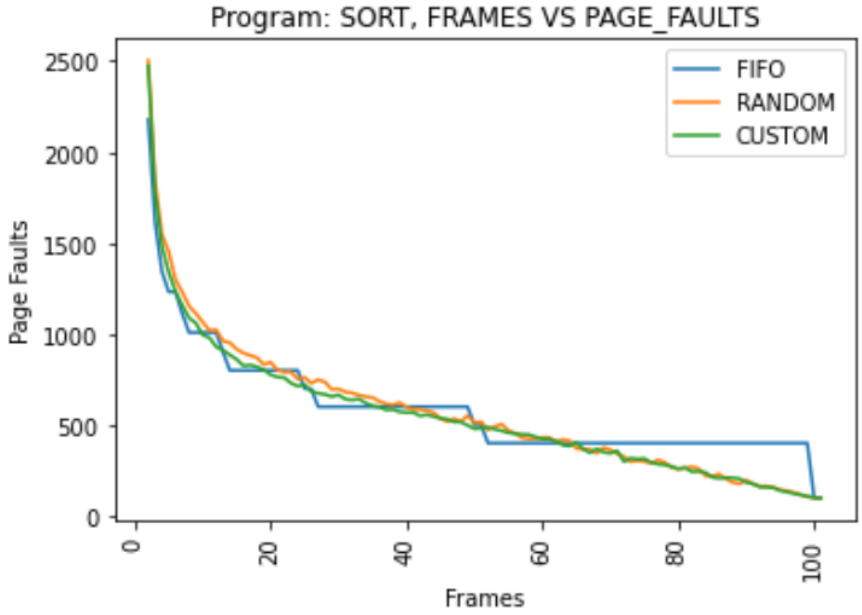


Algorithm: CUSTOM, Program: FOCUS, (Frames VS Page_faults) && (Frames VS disk_writes)



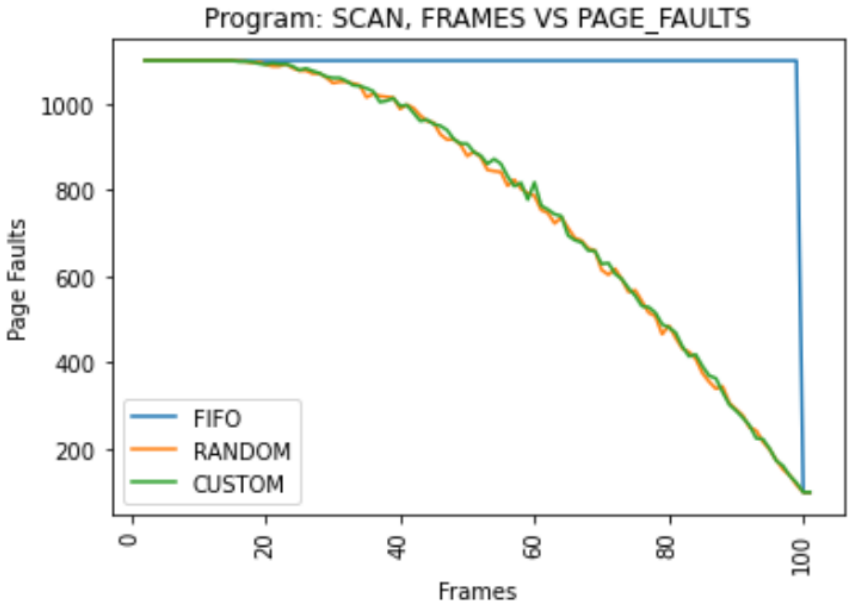
COMPARISON OF 3-PAGE REPLACEMENT ALGORITHMS for the same program:

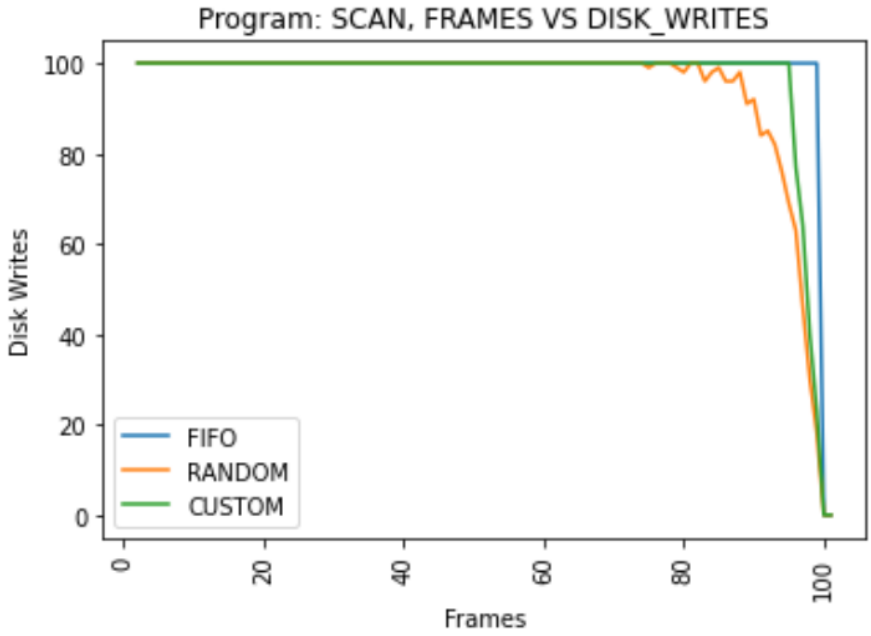
In all the cases, when the number of frames = number of pages, initially page faults occur until all the faces are brought to the memory (i.e 100-page faults to load the data to the memory. Now that all the pages are in the memory page fault will not occur for any subsequent page access. And also there will be no need to perform any disk writes in this case as we are not replacing any pages. A similar pattern will be seen in all the page replacement algorithms when the number of frames = number of pages.



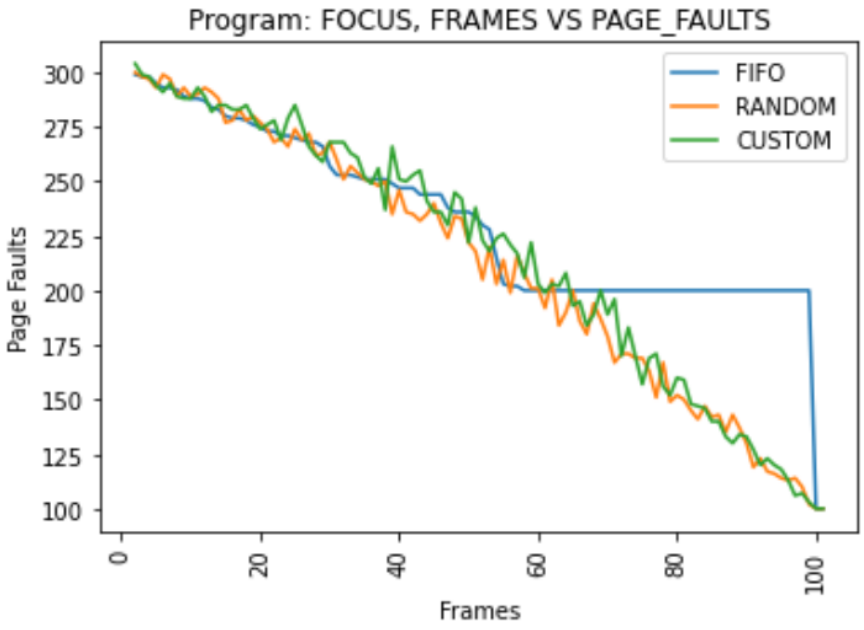
The number of pages is fixed = at 100.

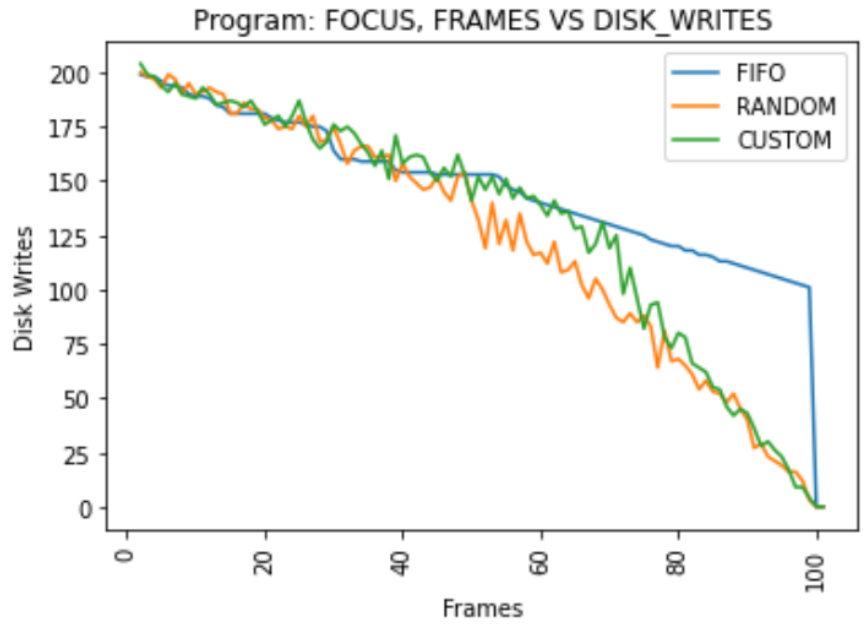
For the sort program, In terms of both page faults and disk writes all three algorithms follow a similar pattern. As the number of frames increases, both the page faults and disk writes decrease gradually as there are more frames to allocate pages. For both page faults and disk writes with less number of frames FIFO is performing better because of the way quick sort works, as once the elements are swapped in an iteration they will not be accessed until the next iteration, so using FIFO works better in this case. But as the number of frames increases random and custom algorithm are performing better.





For the scan function, FIFO performs the worst in terms of both page faults and disk writes because of the way pages are accessed. As the pages are accessed sequentially, the pages that are removed in the FIFO fashion will be needed again in the future, therefore always leading to a page fault. Whereas the other two follow a similar pattern in terms of both page faults and the disk writes. The page faults are less in these algorithms when compared to FIFO because as mentioned earlier for every page access there will be a page fault. Whereas in the other two since pages are removed randomly there might be page fault always. From the graphs, we see that when the frames are more the number of disk writes is less in the case of custom and random. Disk writes occur when the page with a write bit set is replaced from the physical memory when the number of frames is more because of the way pages are replaced in these two algorithms some pages with a write bit set may not be replaced.





In the focus algorithm, initially, all the pages are accessed and then random parts of the array are selected and elements in that random part are updated at random positions. Since the parts are selected at random and each random part size is less than the page size, so all the algorithms perform the same for this. So as you can see in the graphs all three algorithms are following a similar pattern. However in the above graph, we can see that from a certain number of frames the page faults are fixed to 200 because the first 100 page faults are due to the initial for loop for all accessing the elements once, and later if the page fault occurs in the focus part of the code then again while accessing elements after the focus loop, the page fault will not occur as this page, as this page is now in the memory because of the way fifo works. So in total 100-page faults occur for the later part of the code. For the focus program disk writes will follow the same pattern as page faults, as whenever a page fault there is a chance that a page fault can also occur.