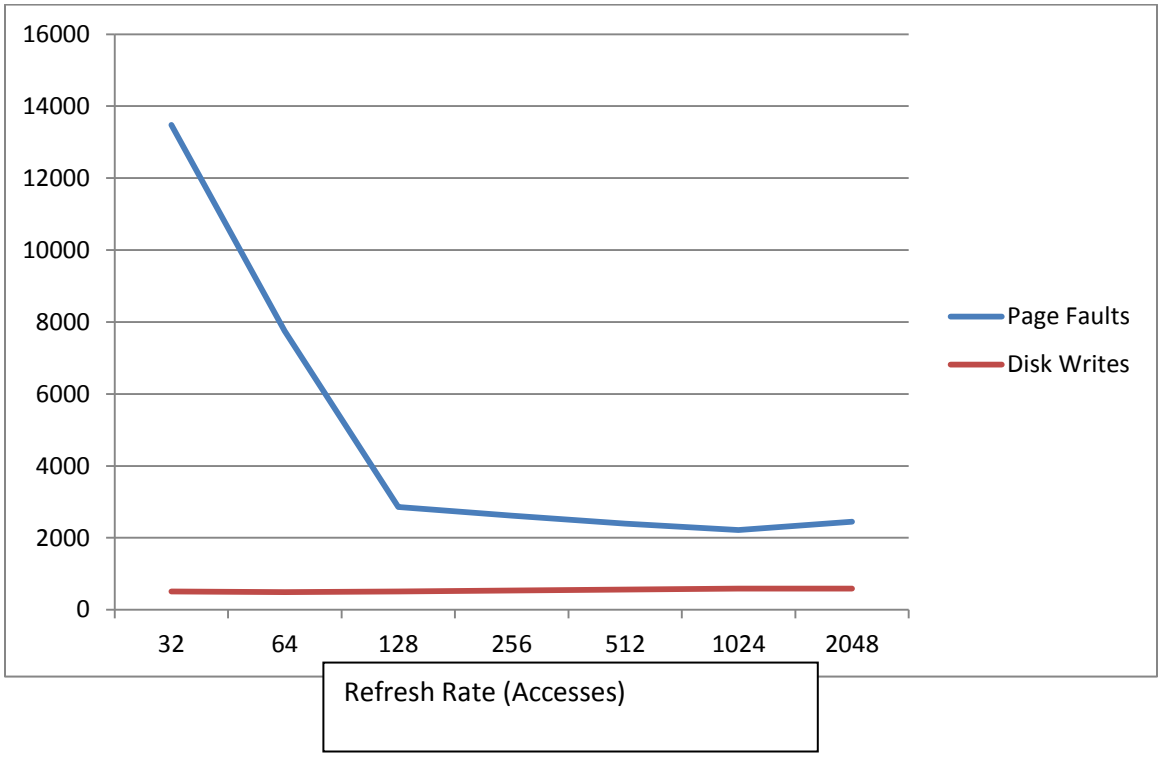
Adam Jaworski

CS 1550

Project 3 Write-Up

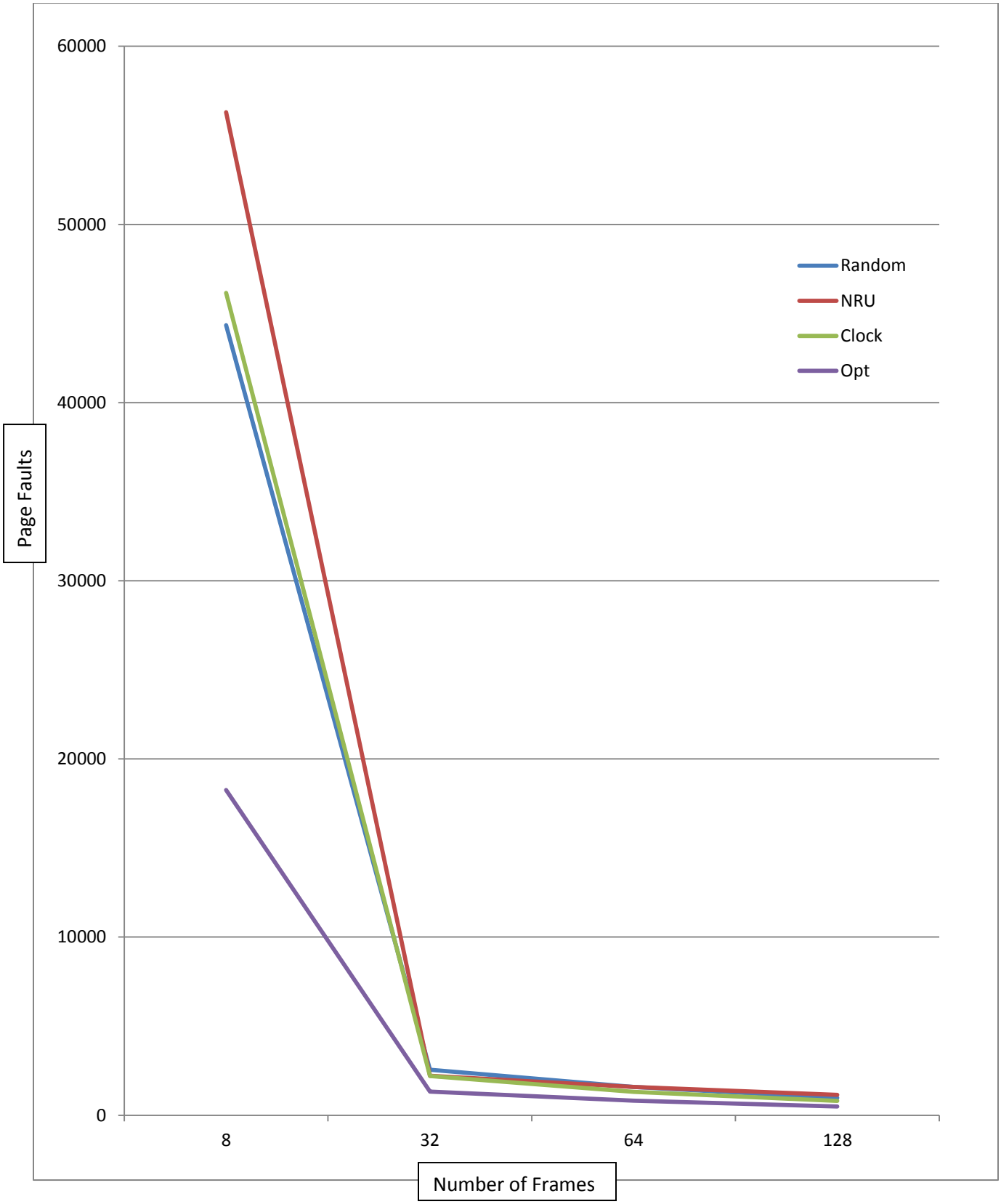
Determining NRU Refresh Rate



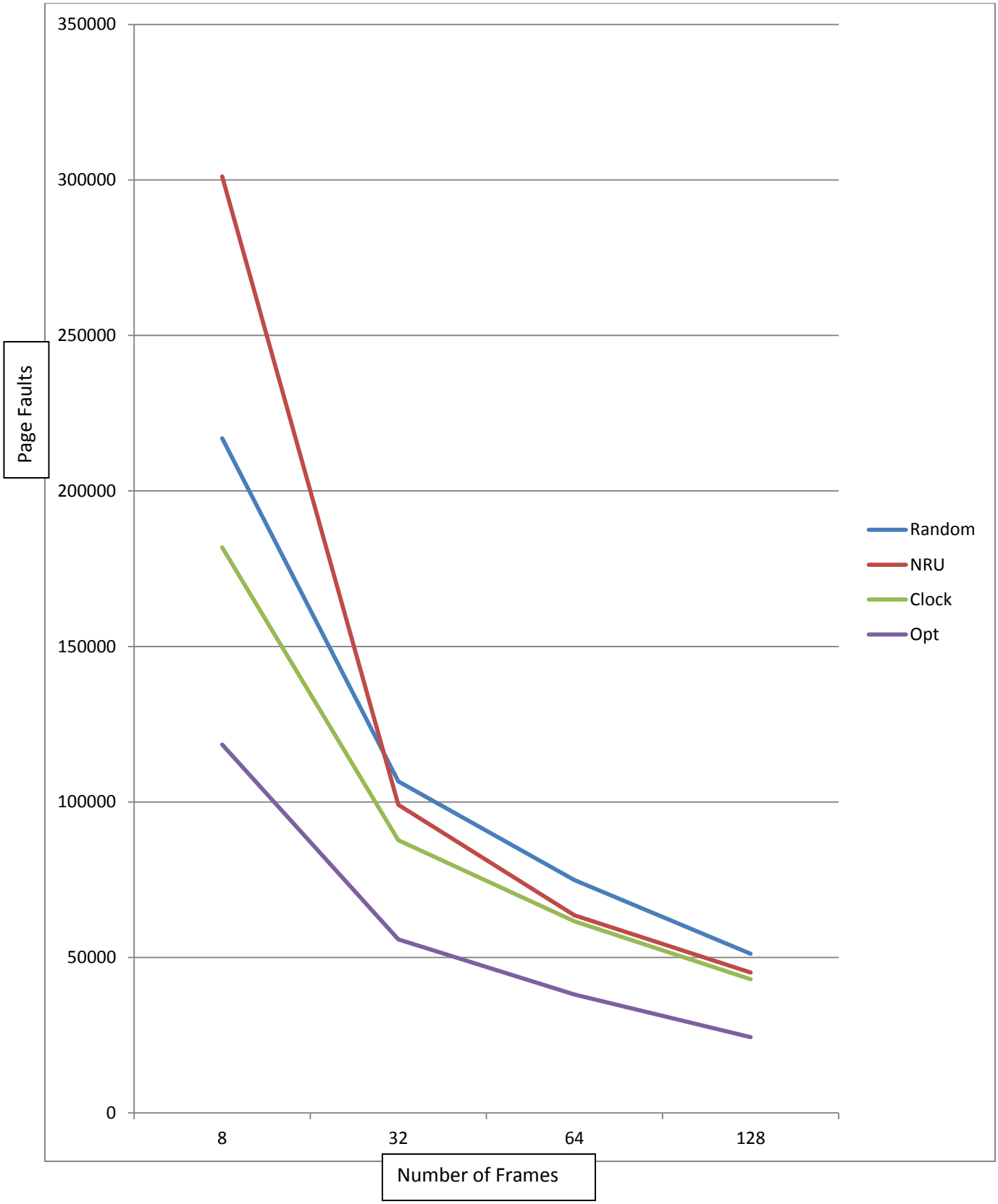
In determining how many references to wait between refreshes, I tested on the bzip.trace file with 32 frames. I decided to start by doing a refresh every 32 references, and then doubling the spacing between refreshes until I stopped seeing a reduction in the number of page faults. As the graph illustrates, this point was reached at a refresh rate of 2048, at which point the number of page faults started to go back up. Therefore, I selected the refresh rate before that (1024) as a good rate to use.

	32	**64**	**128**	**256**	**512**	**1024**	**2048**
Page Faults	13482	7749	2858	2620	2395	2217	2445
Disk Writes	505	490	511	538	565	587	593

Bzip.trace


60000
50000
40000
30000
20000
10000
0
Page Faults
Random
NRU
Clock
Opt
8
32
64
128
Number of Frames

Gcc.trace


350000
300000
250000
200000
150000
100000
50000
0
Page Faults
8
32
64
128
Number of Frames
Random
NRU
Clock
Opt

Bzip.trace (red = page faults, brown = disk writes)

	8	32	64	128
Random	44344, 16430	2551, 876	1588, 559	972, 334
NRU	56303, 2937	2217, 587	1583, 313	1142, 71
Clock	46164, 17568	2203, 734	1318, 443	802, 235
Opt	18251, 7581	1330, 459	821, 284	497, 139

Gcc.trace (red = page faults, brown = disk writes)

	8	32	64	128
Random	216908, 37396	106587, 17231	74904, 12170	51167, 8609
NRU	301133, 14793	99066, 9579	63576, 7684	45149, 4132
Clock	181856, 29401	87686, 12293	61640, 9346	43017, 6671
Opt	118480, 15031	55802, 8278	38050, 5741	24391, 3980

As expected, my results indicated that the number of both page faults and disk writes goes down with an increasing number of frames. However, the number of faults will still asymptotically approach some lower limit (the number of compulsory misses) no matter how many frames are available. In this way, the differences between the various algorithms become less important as the number of available frames is increased. This was particularly evident with the bzip trace file.

One thing that stands out to me is that sometimes the NRU algorithm would yield more page faults than just evicting a page at random. I think the most likely cause of that is looping. For instance, if x frames are available and a loop references x+1 pages exactly once on each iteration, then it may be that the algorithm would be protecting pages “behind” it while leaving pages in “front” of it vulnerable to eviction. In this way, it would actually have a preference for evicting pages it will need very soon.

Another thing that stands out to me from the results I obtained is that the NRU algorithm consistently produced the lowest numbers of disk writes (opt excluded, naturally), even when it generated more page faults (and therefore evictions) overall. This was not that surprising to me however, as the NRU algorithm was the only one to take dirty bits into consideration at all (preferring to evict clean pages over dirty ones, thus avoiding disk writes).

Naturally none of the three implementable algorithms tested were able to match opt’s performance. However, opt isn’t an option in real-world scenarios, and so therefore for an actual operating system the best algorithm is the one that comes the closest to opt’s ideal performance. The clock algorithm had the best performance in terms of total numbers of page

faults, NRU had the best performance in terms of disk writes, and the random algorithm had the best performance in neither (and in fact was surpassed by the clock algorithm in both metrics). Therefore the choice seems to be between clock and NRU. Based on my results, I would ultimately have to recommend the clock algorithm, as I think it would be more important to minimize total page faults over disk writes (since I think the disk should be able to do some of its work in parallel anyway). NRU's apparent issues with looping also seem potentially problematic.