CMPUT 379 Assignment 3

Michael Kwok

April 15, 2021

All of the data was collected by the following command:
`valgrind --tool=lackey --trace-mem=yes <program> 10000 2>&1 | ./valws379 <pgsize> <windowsize>`

The first set of references in the beginning of the graphs are similar for all the data, explained by the fact that they all set up the sorting array similarly. At the end of the program, a large spike appears as allocated items get freed and cleaned up.

The 4096 and 8192 byte page graphs are similar in shape, but the 8192 graphs are lower as each page encompasses more references, and thus count for less "pages" loaded.

In the following sections, the 4k page size results will be focus, as they show more extreme differences between input sizes, and the same explanations can be applied to the 8k page results. Similar behaviour such as the beginning and the end was explained previously and will not be repeated below.

The following timing results were found for each sorting algorithm:

Sorting Algorithm	Time (s)
Radix Sort	36.482
Quick Sort	25.335
Heap Sort	39.481

In the following sections, it can be seen that Heap Sort has the most memory accesses, Radix Sort having the 2nd most and Quick Sort has the least number of accesses, which explains the time difference.

1 Heap Sort

The first section of both graphs show the heapify section of the heapsort process. Since heapify has to go through the entire array to swap and build the binary tree, the number of pages required for it will increase as time passes. After building the heap is done, the test code calls `fprintf`, which explains the massive spike in the middle, as the program was linked dynamically. After the `fprintf`, the working set starts a decreasing trend. This looks higher in the 100 000 windowsize case, as it still has references from the previous parts of the program in the "working set". As time goes on, the number of pages kept in the working set gets dropped as it goes out of the sliding window before stabilizing at the end of sorting.

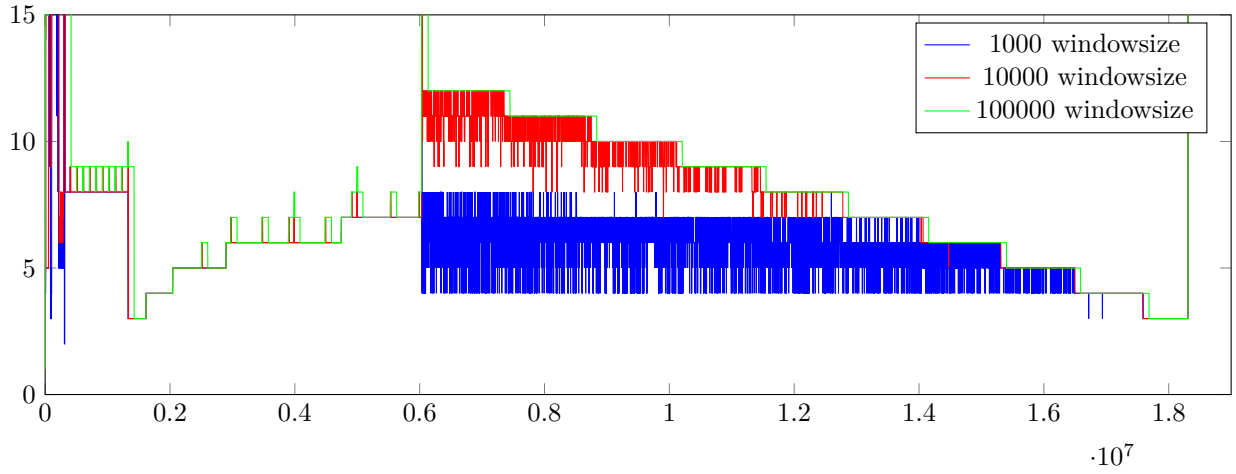

Figure 1: Heap Sort with 4096 Page Size

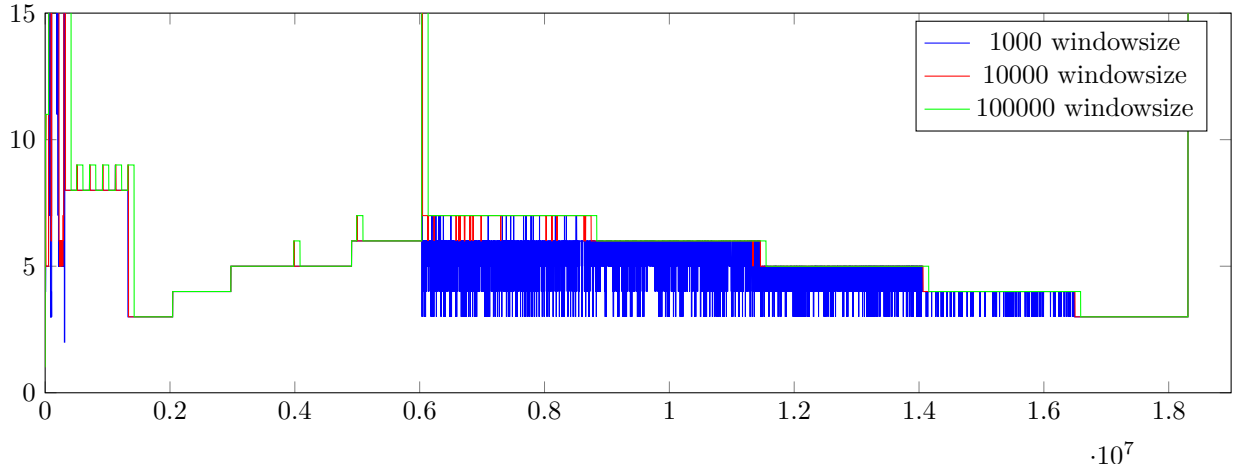

Figure 2: Heap Sort with 8192 Page Size

2 Radix Sort

The graph shows the cyclic fashion of radix sorting. The entire array gets traversed in each cycle, as the way radix sort works is by sorting the digits of each entry relative to each other. During the cycle iterations, less and less of the array is looked at as they slowly get sorted in the relative order. As shown with the 1000 case, there are 10 peaks, and each plateau is a single digit in the number.

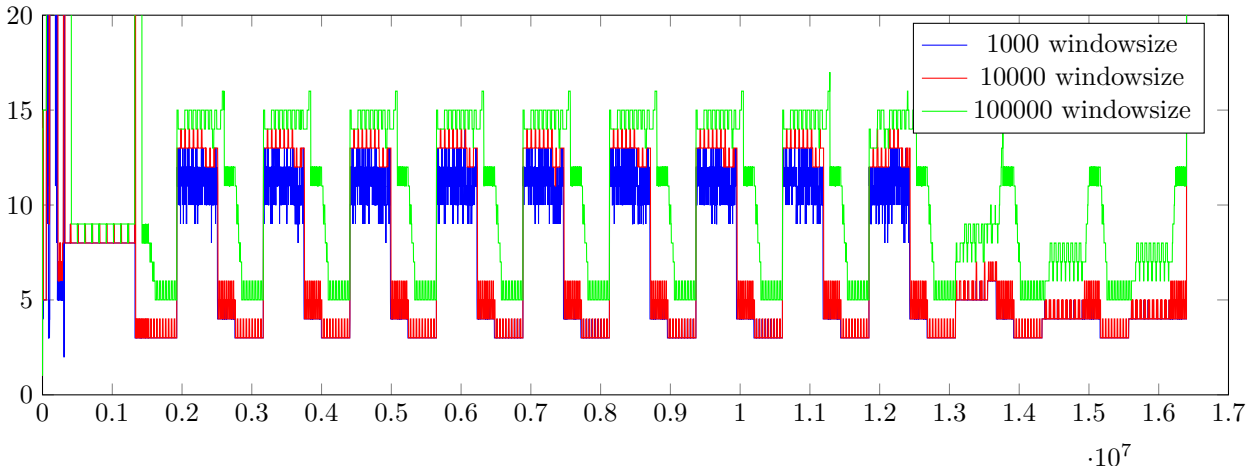

Figure 3: Radix Sort with 4096 Page Size

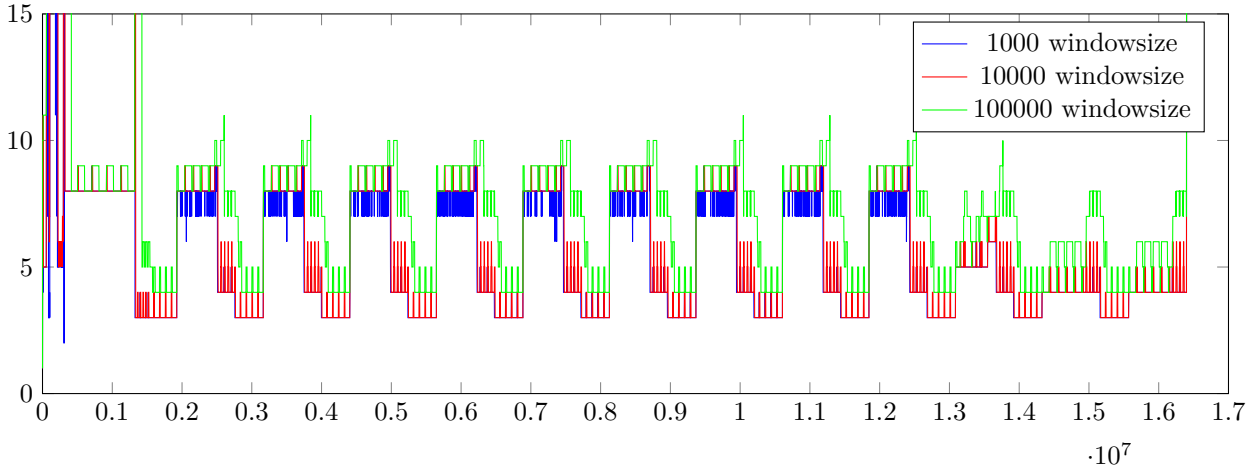

Figure 4: Radix Sort Graphs with 8192 Page Size

3 Quick Sort

For quicksort, the sorting algorithm works on small blocks of the array at a time, which is explained in the lack of pages in the working set for most of the smaller window sizes. The larger window size shows that the working sets are not overlapping, showing up as more pages compared to the smaller window sizes, as it "remembers" a longer reference access history.

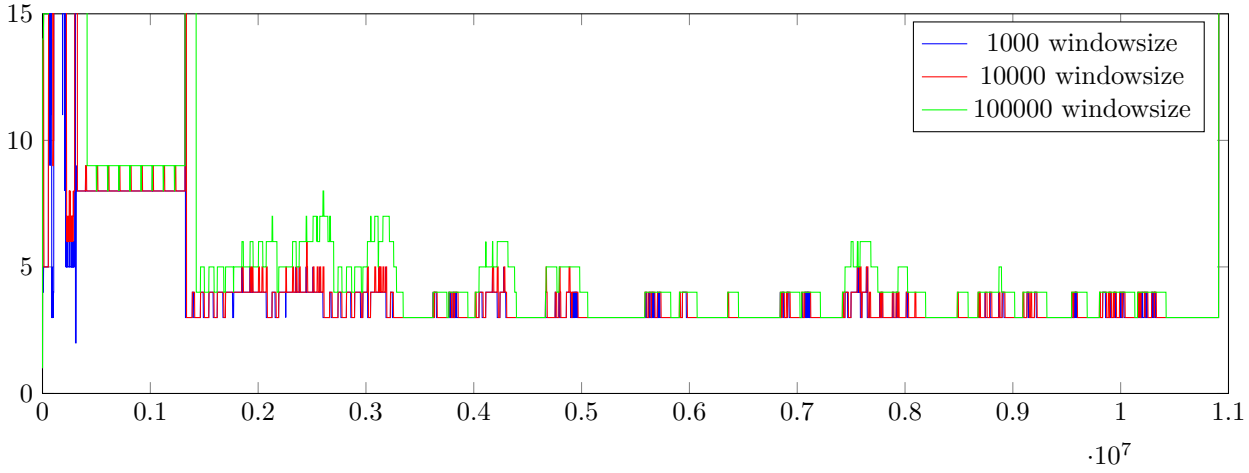

Figure 5: Quick Sort Graphs with 4096 Page Size

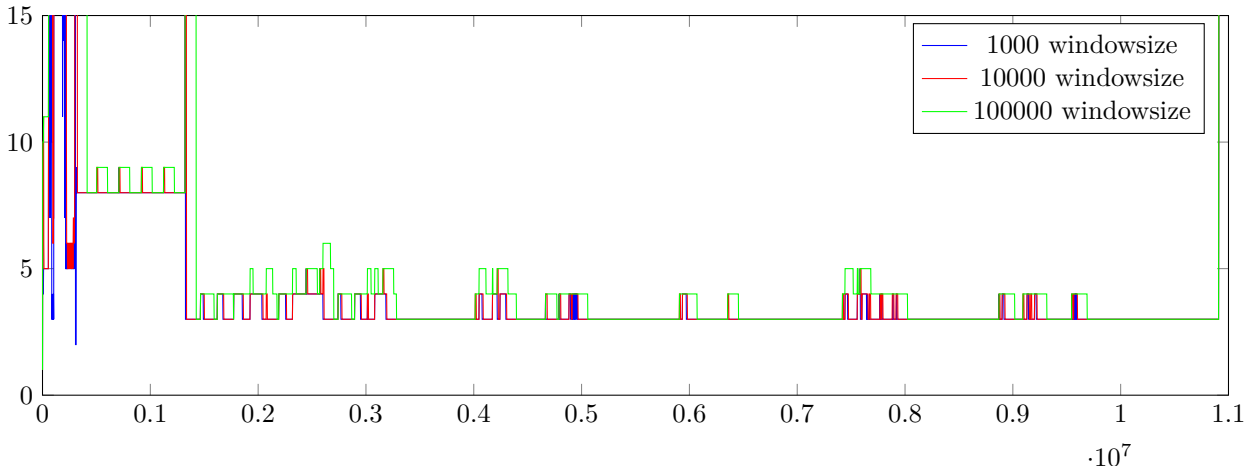

Figure 6: Quick Sort Graphs with 8192 Page Size