Allan Manuba

Performance Study

1 Heapsort

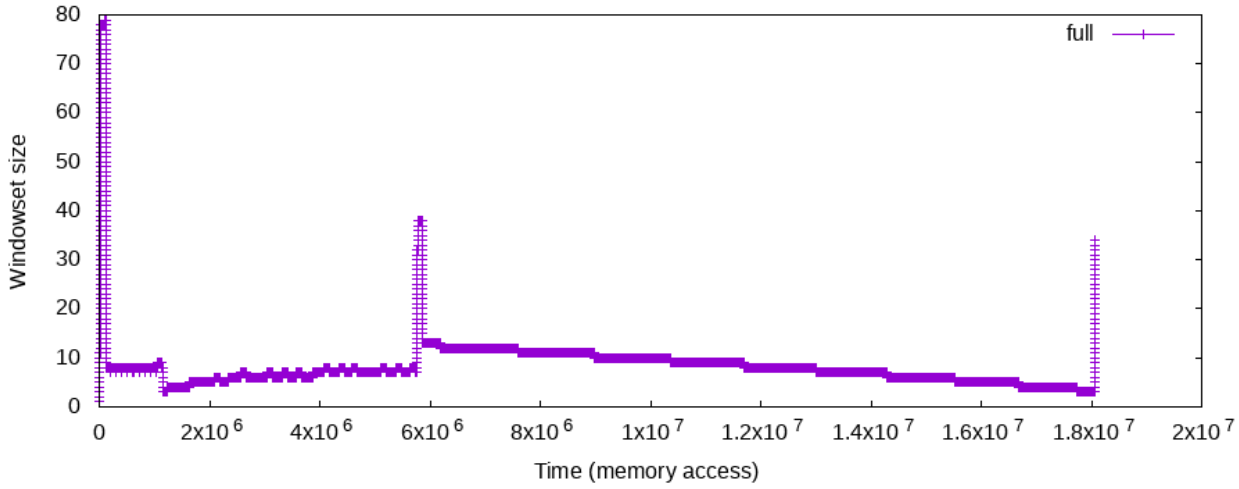

Figure 1.1 Heapsort (data size=10 000; skipsize=0; page size=4 096; window size=100 000)

Here is the full graph for heapsort. For the following setups, we want to find the interesting parts, and cut-off parts that are not interesting. There are two interesting parts in heapsort: heapify and heapsort itself. Instead of manually finding out the skipsize, I have added a functionality to valws379 that automatically trims the data to the interesting point. The added functionality trims both the leading data and the trailing data.

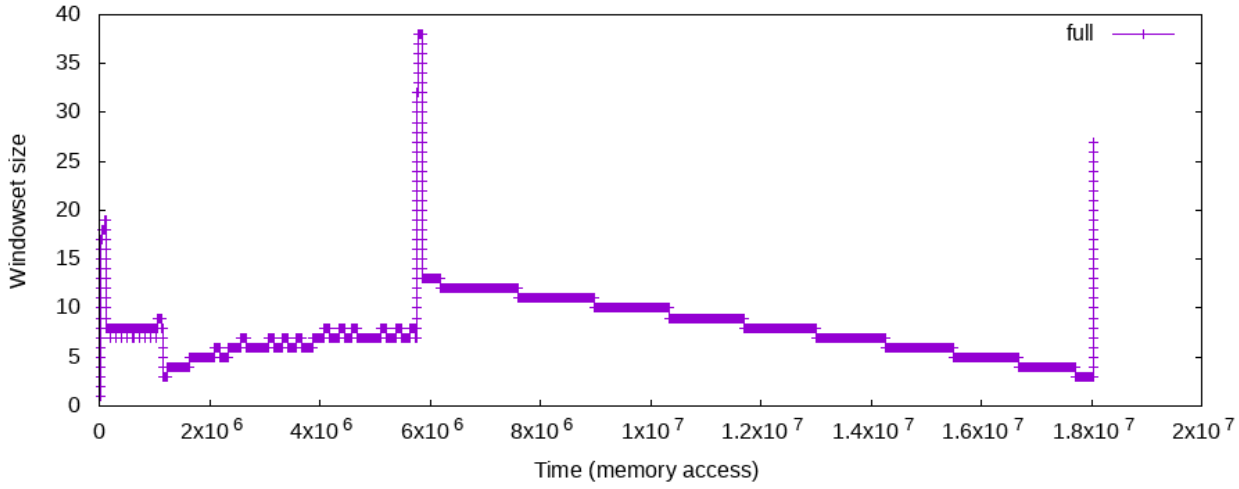

Figure 1.2 Heapsort (data size=10 000; skipsize=32 575; page size=4096;

window size=100 000)

This is the interesting section. This is where both Heapify and Heapsort happens. It would help a lot if we know where's where.We see the spike reaching 80 gone when it starts here.

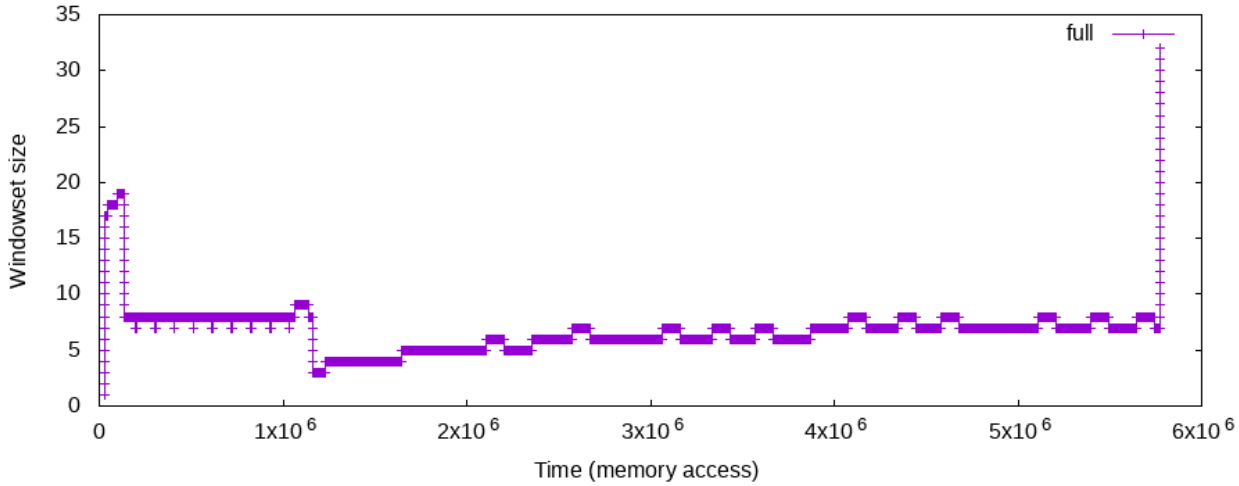

Figure 1.3 Heapsort (data size=10 000; skipsize=32575; page size=4096; window size=100 000)

This part is where Heapify happens.

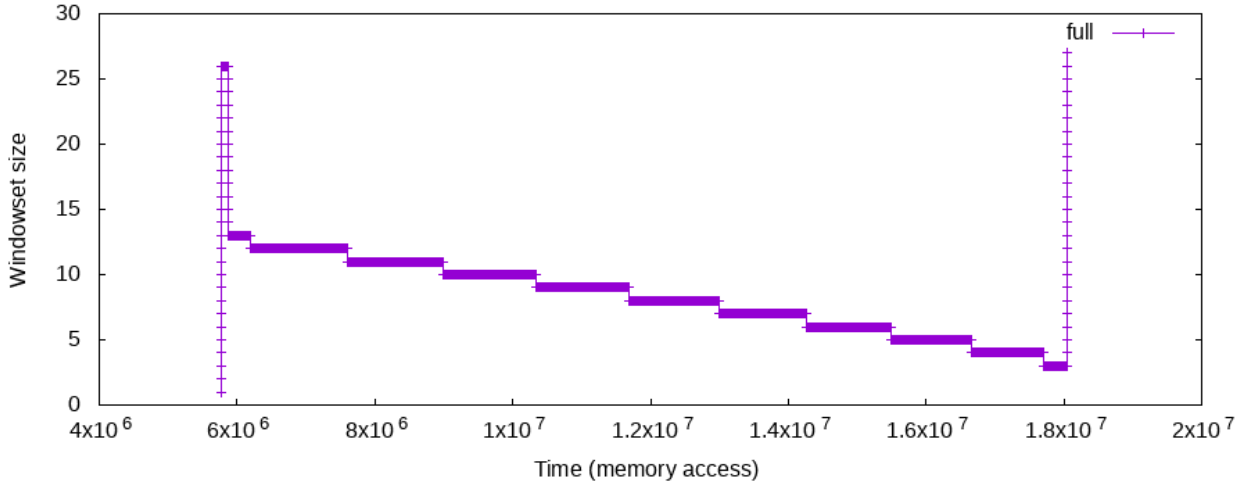

Figure 1.4 Heapsort (data size=10 000; skipsize=5 768 885; page size=4 096; window size=100 000)

This part is where Heapsort happens. We start to notice a pattern where the spike occurs where we enter a function. We may speculate that this is due to new instructions that may be related to initialization and clean up.

Now, let's try adjusting the page size.

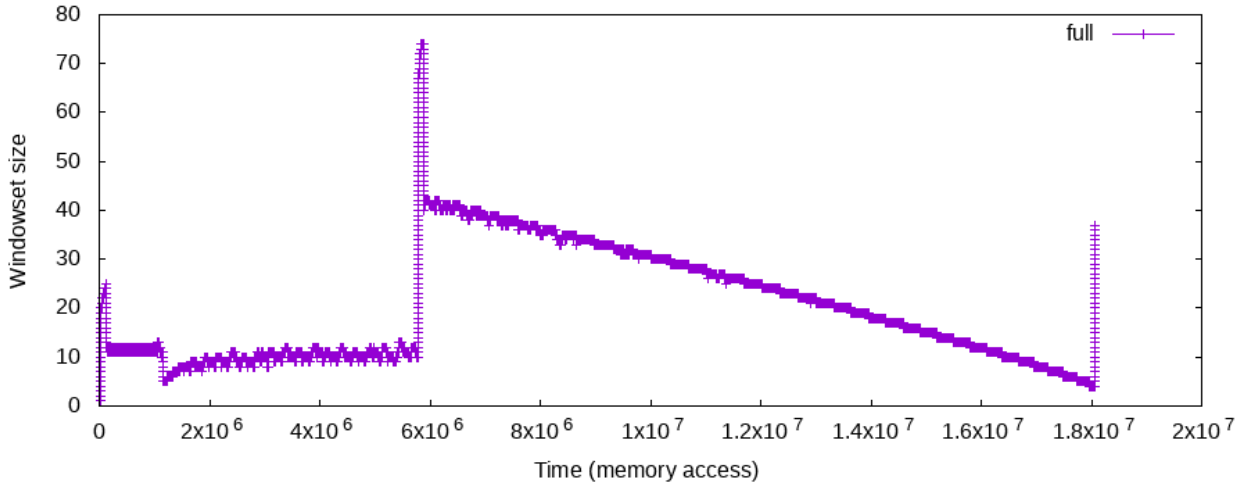

Figure 1.5 Heapsort (data size=10 000; skipsize=32 575; page size=1024; window size=100 000)

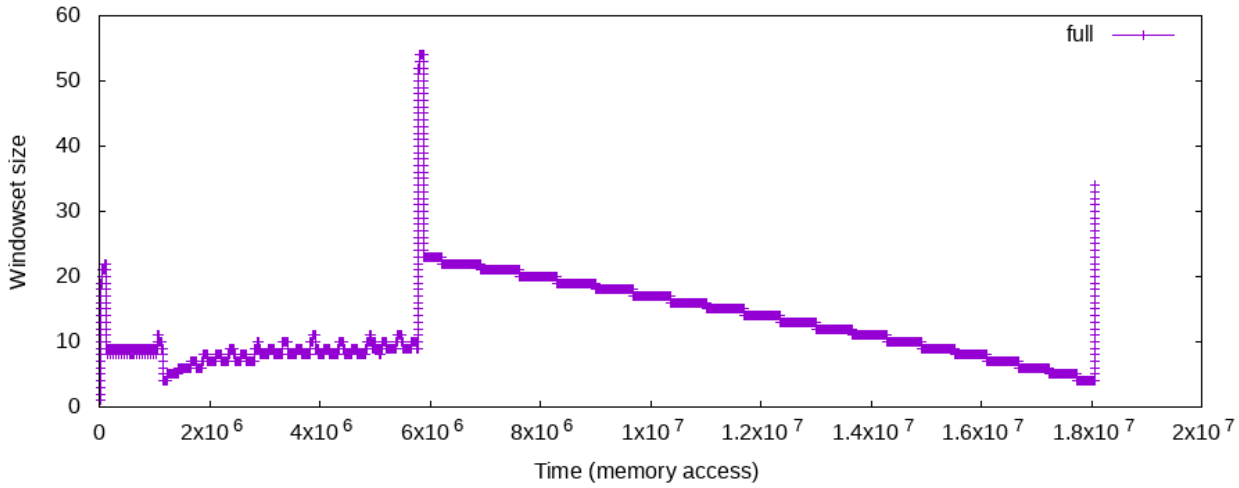

Figure 1.6 Heapsort (data size=10 000; skipsize=32 575; page size=2 048; window size=100 000)

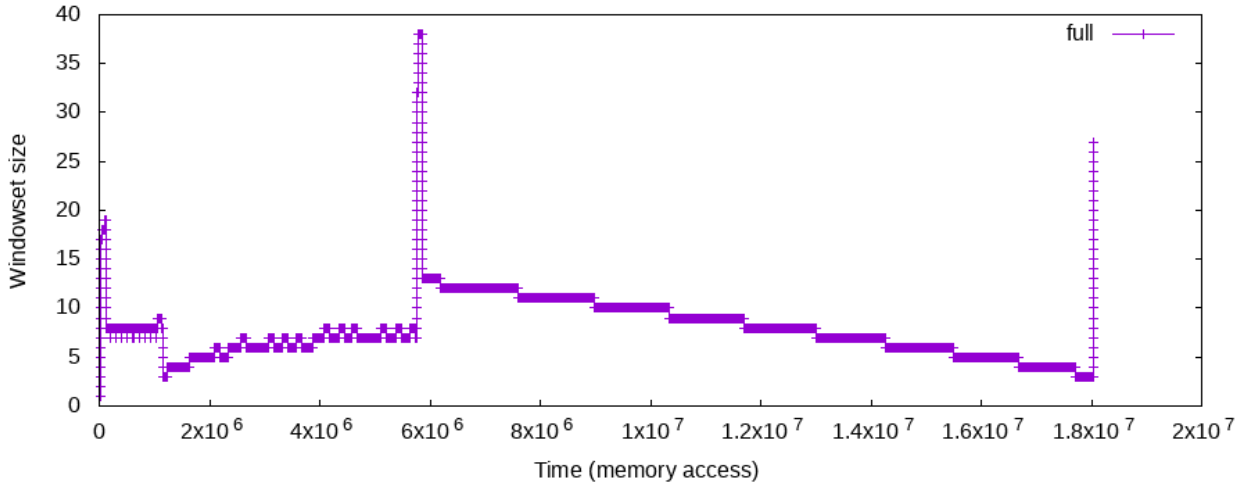

Figure 1.2 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=100 000)

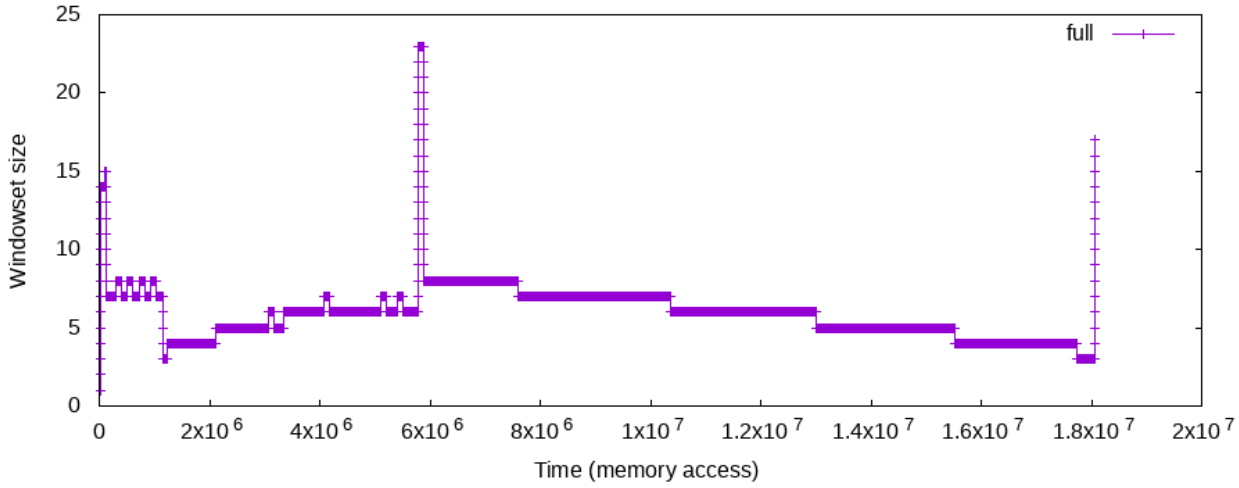

Figure 1.7 Heapsort (data size=10 000; skipsize=32 575; page size=8 192; window size=100 000)

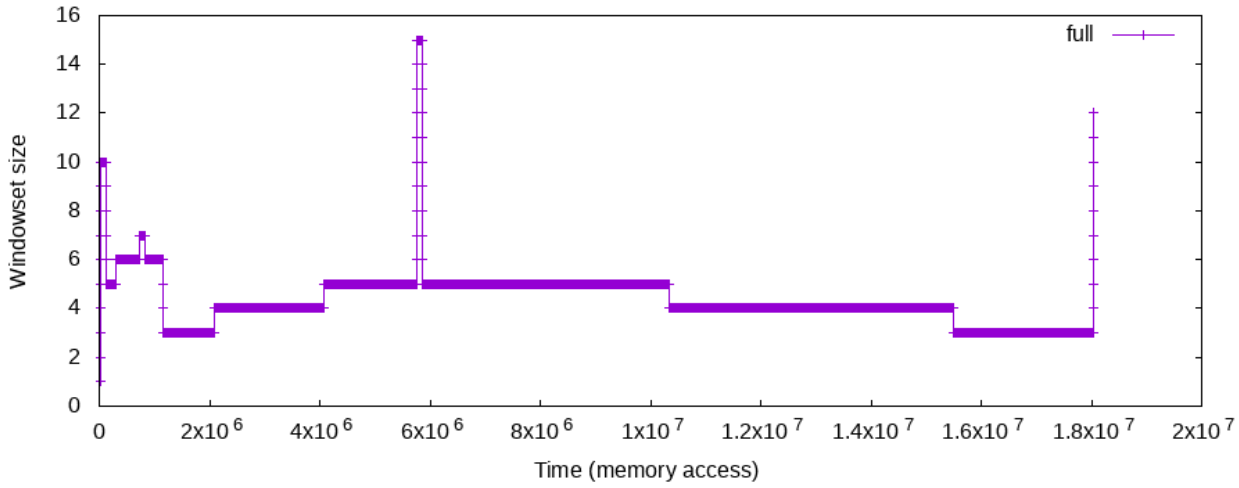

Figure 1.8 Heapsort (data size=10 000; skipsize=32 575; page size=16 384; window size=100 000)

There seems to be a trend where the overall windowset size decreases as we increase the page size.

Let's try observing how the graph changes when we change the input data size.

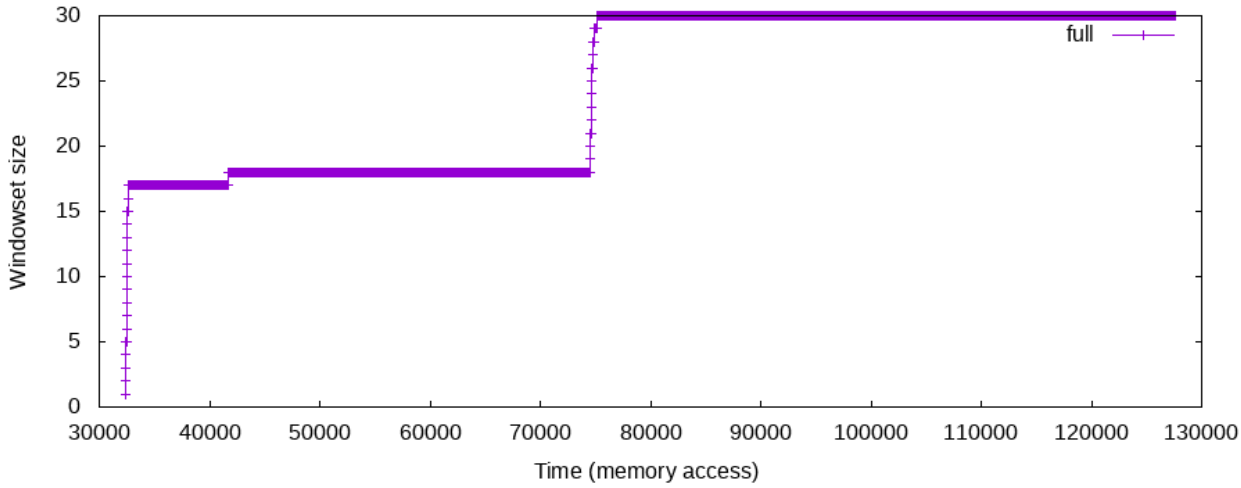

Figure 1.9 Heapsort (data size=100; skipsize=32 575; page size=4 096; window size=100 000)

With a low input data size, we could hardly tell the spikes that separates Heapsort from Heapify.

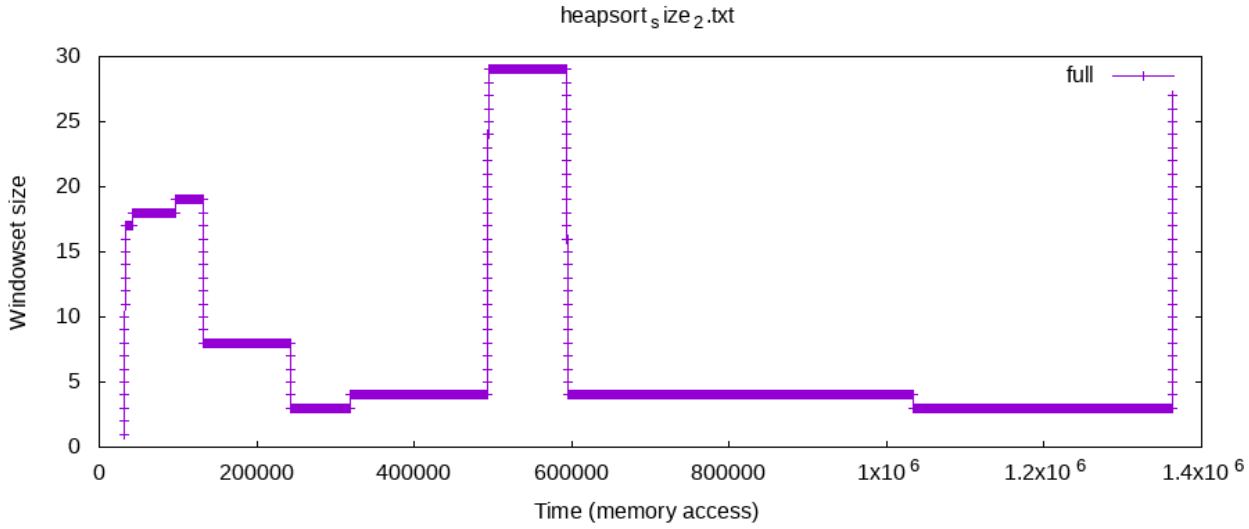

Figure 1.10 Heapsort (data size=1 000; skipsize=32 575; page size=4 096; window size=100 000)

We start to notice the spike pattern that differentiates the part where Heapsort and Heapify happens.

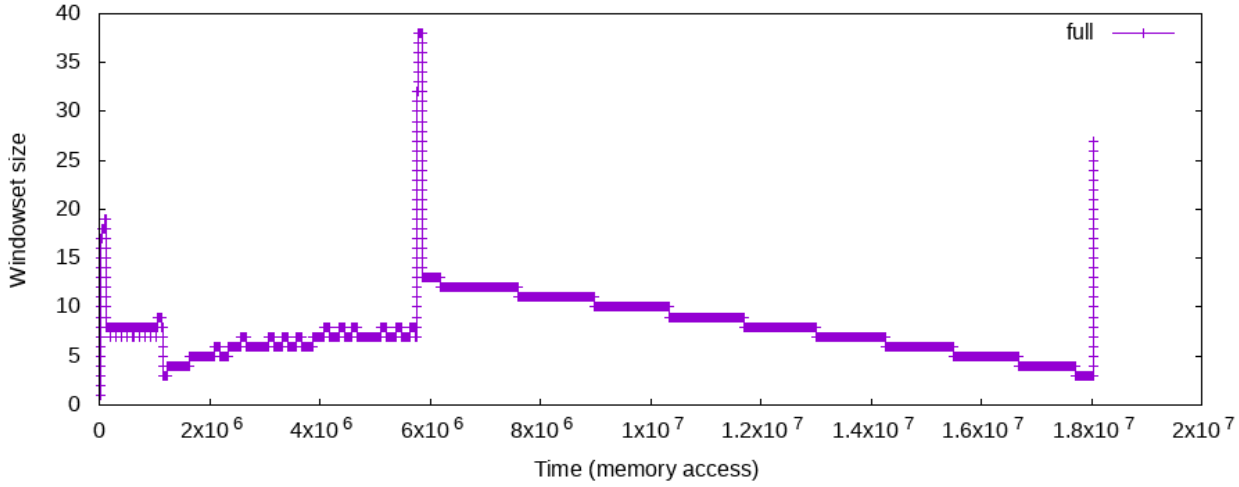

Figure 1.2 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=100 000)

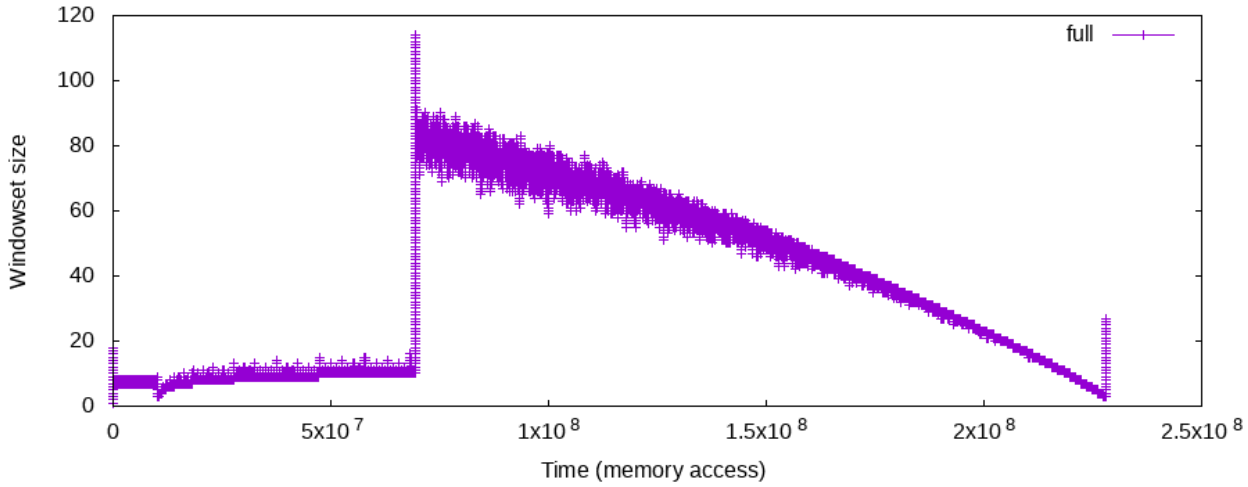

Figure 1.11 Heapsort (data size=100 000; skipsize=32 575; page size=4 096; window size=100 000)

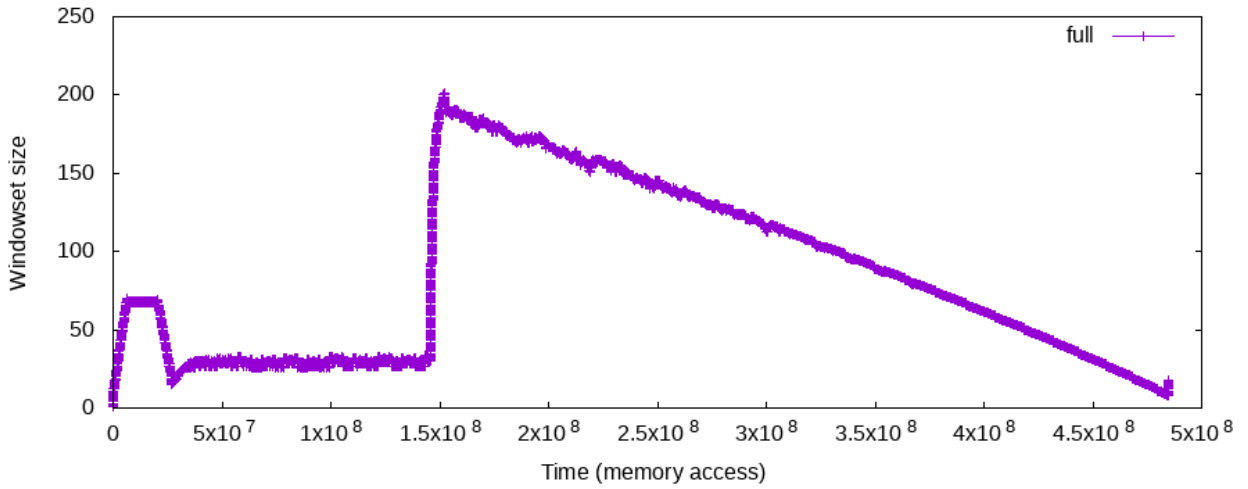

Figure 1.12 Heapsort (data size=200 000; skipsize=32 575; page size=4 096; window size=100 000)

As we increase the data size, the procedures in-between the initialization and clean up for Heapsort function starts to catch up with the two spikes. The spike after Heapsort becomes less clear in the bigger data sizes.

We next observe the trends when we modify the window size.

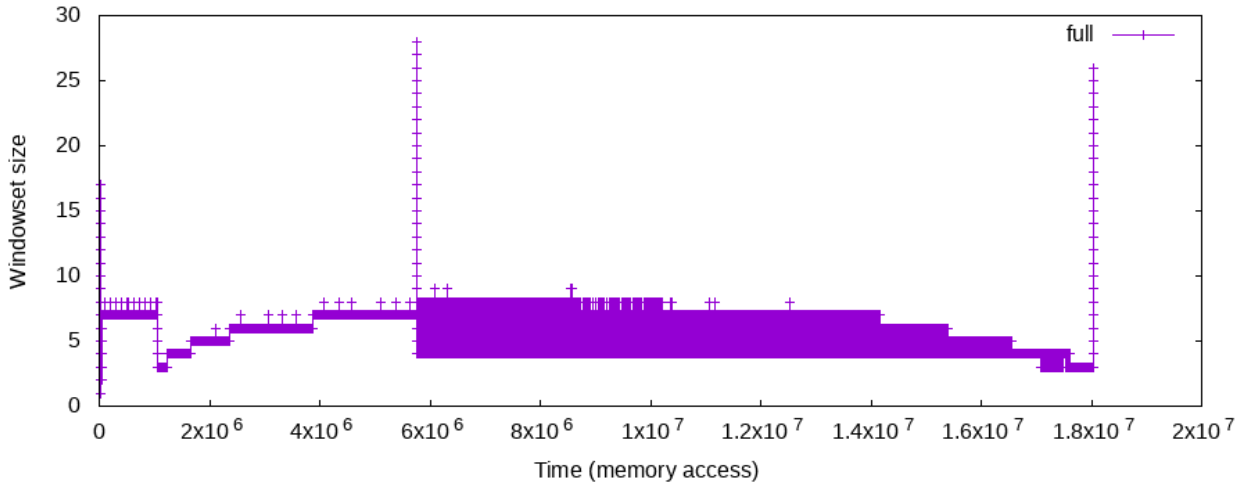

Figure 1.13 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=1 000)

The windowset size seems to go up and down a lot, which creates this jagged lines into this solid color. Then again, the windowset size is affected by window size which explains the small windowset size that we seem be around.

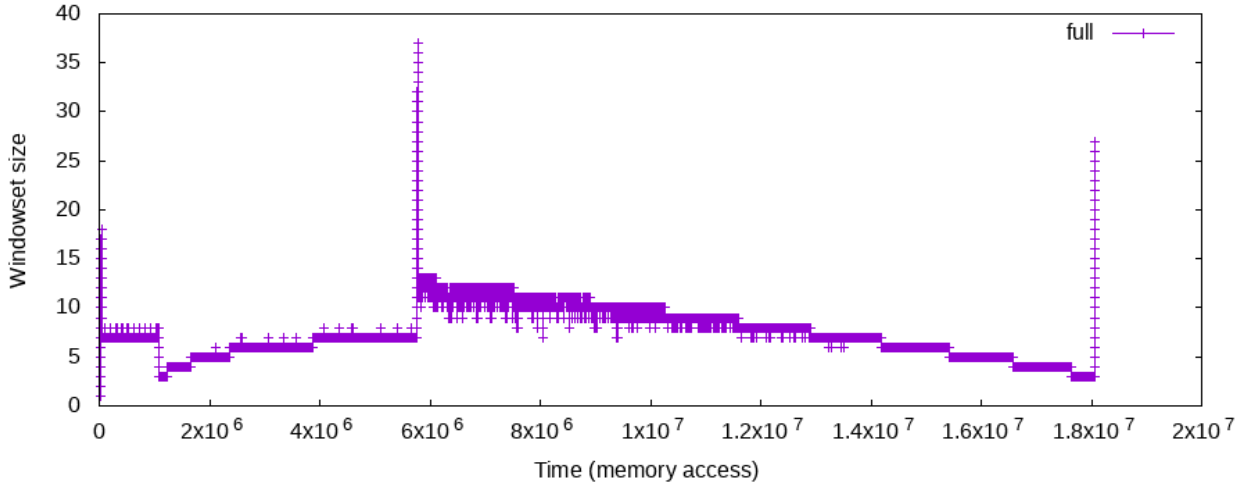

Figure 1.14 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=10 000)

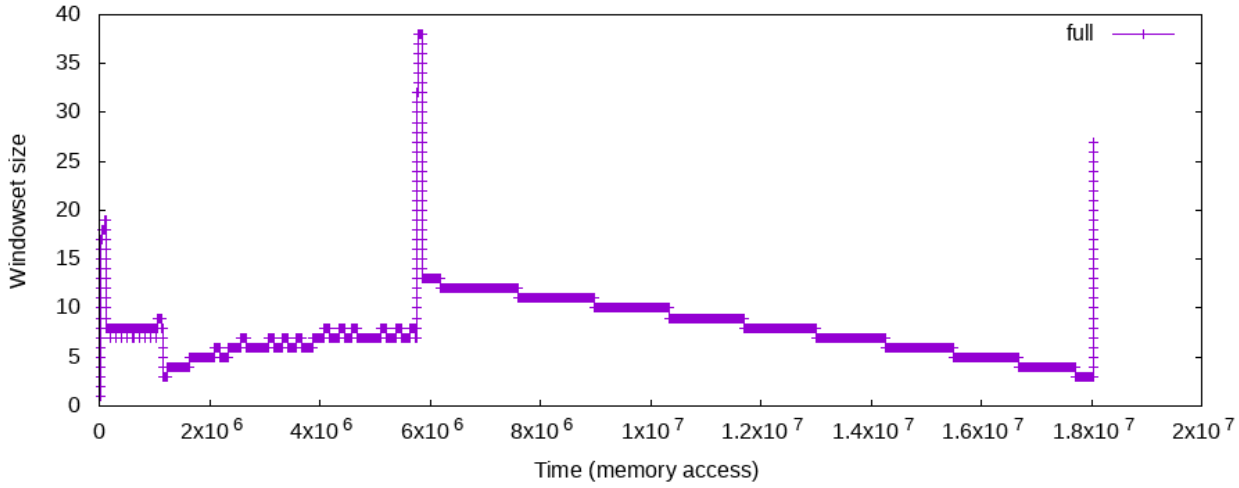

Figure 1.2 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=100 000)

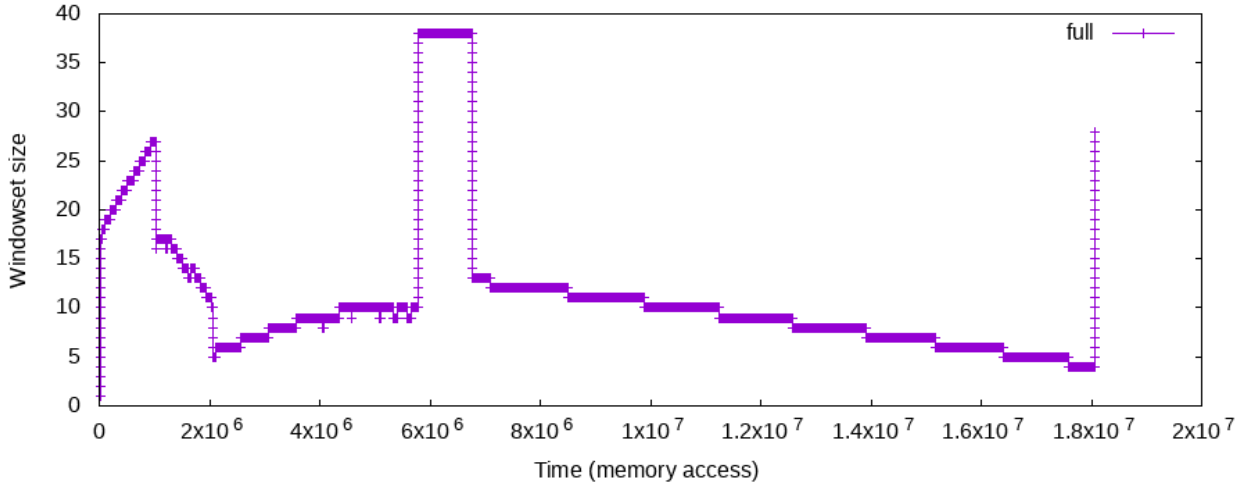

Figure 1.15 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=1 000 000)

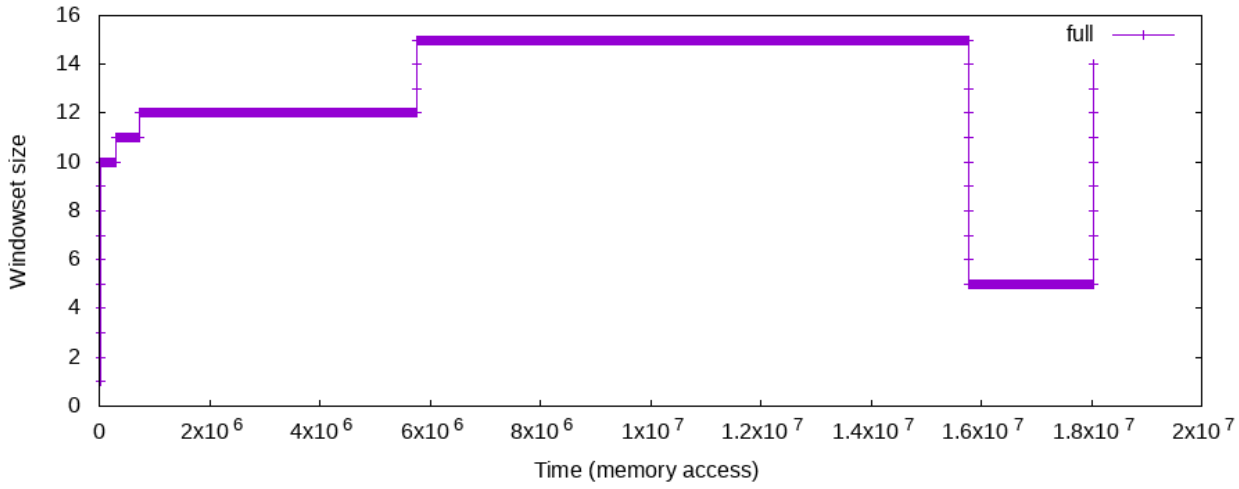

Figure 1.13 Heapsort (data size=10 000; skipsize=32 575; page size=4 096; window size=10 000 000)

The trend that I was expecting for increasing the window size is how the data would plateau. The highest window size used does follow the pattern, but the peak windowset size is suspicious.

An interesting windowset size behavior for Heapsort is how it decreases over time. This may be because our number of elements decreases, and we only access contiguous parts of the data.

2 Quicksort

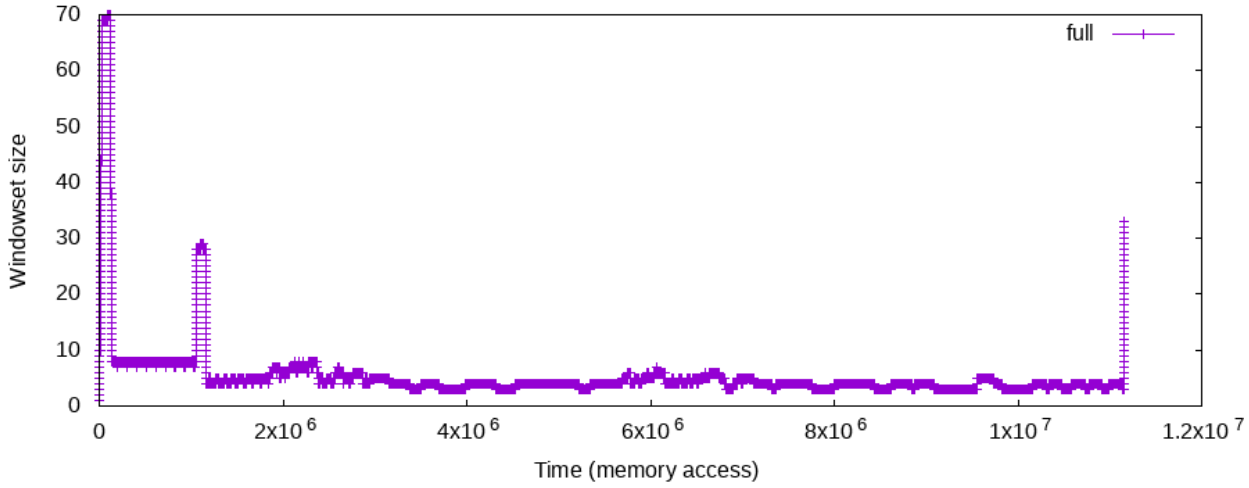

Figure 2.1 Quicksort (data size=10 000; skipsize=0; page size=4 096; window size=100 000)

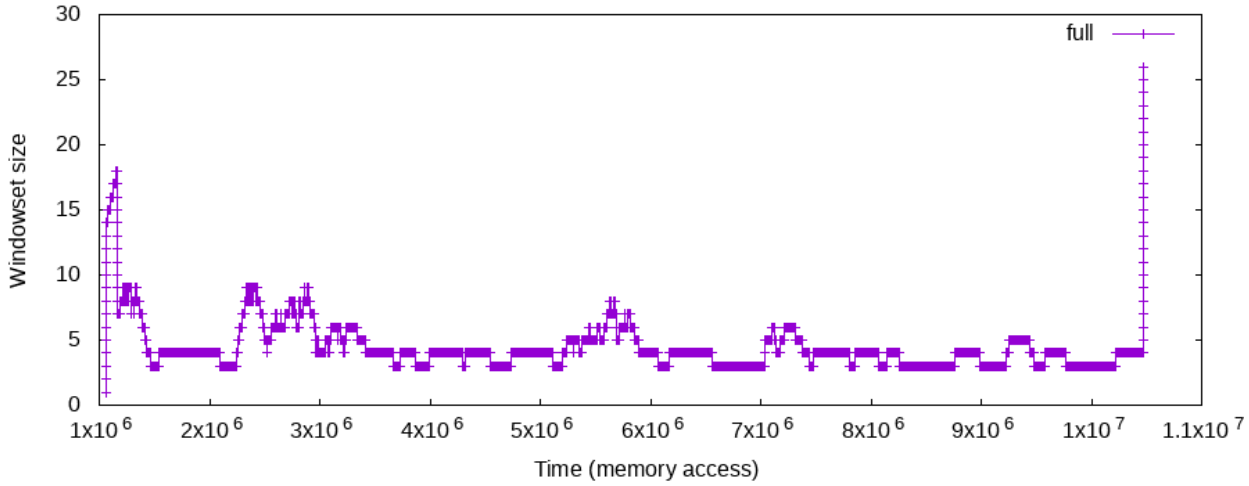

Figure 2.2 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=100 000)

What's interesting with quicksort is how it doesn't seem to have a trend like Heapsort has.

Below is observing Heapsort in increasing page size.

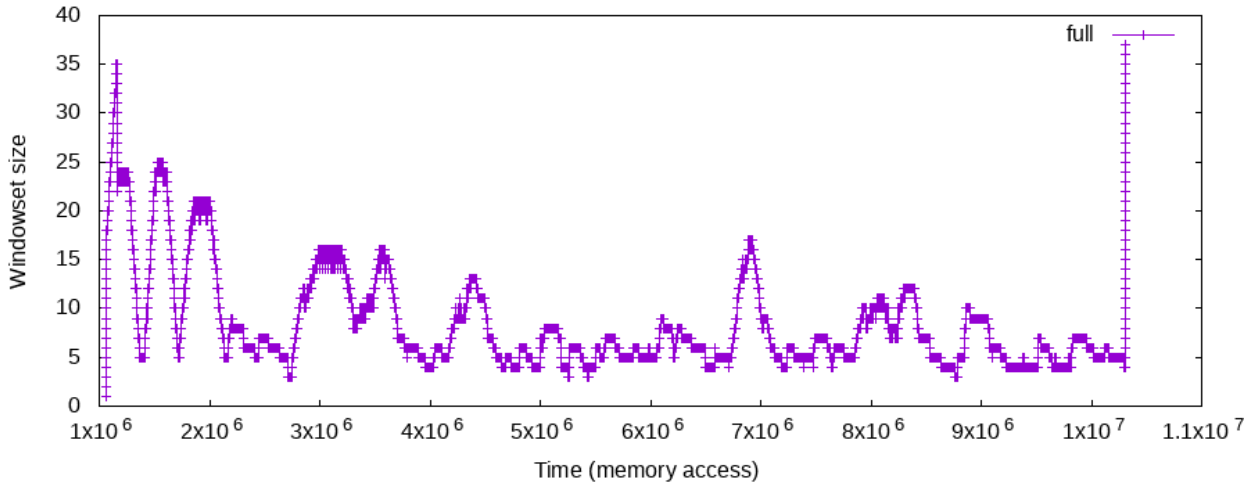

Figure 2.3 Quicksort (data size=10 000; skipsize=1 060 174; page size=1 024; window size=100 000)

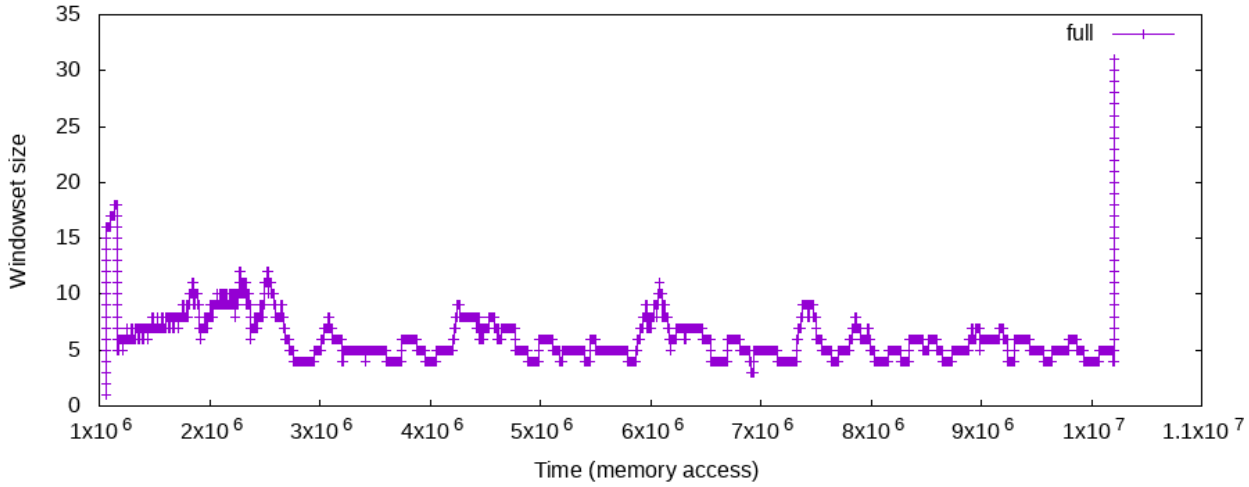

Figure 2.4 Quicksort (data size=10 000; skipsize=1 060 174; page size=2 048; window size=100 000)

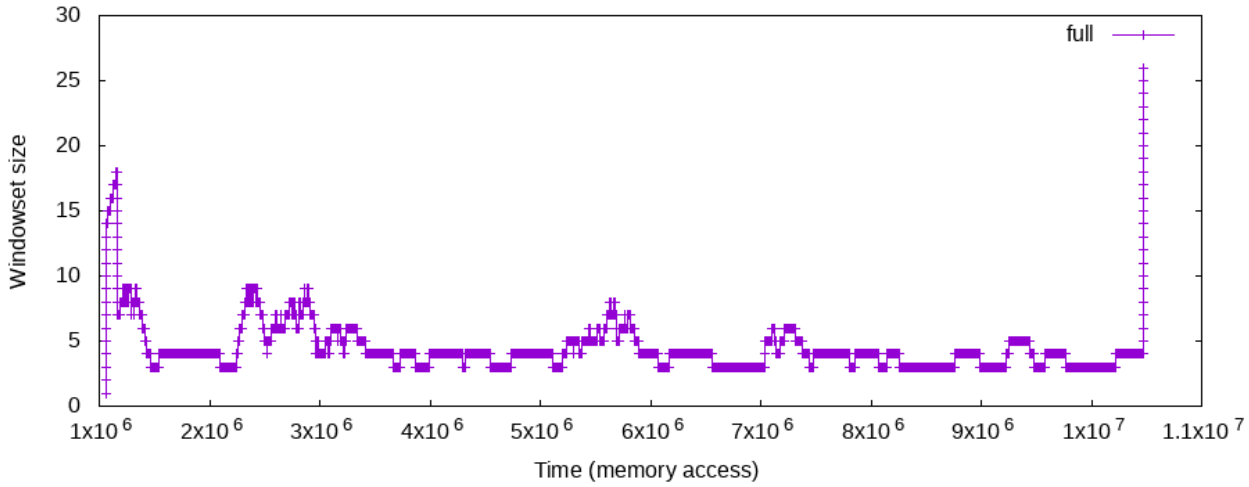

Figure 2.2 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=100 000)

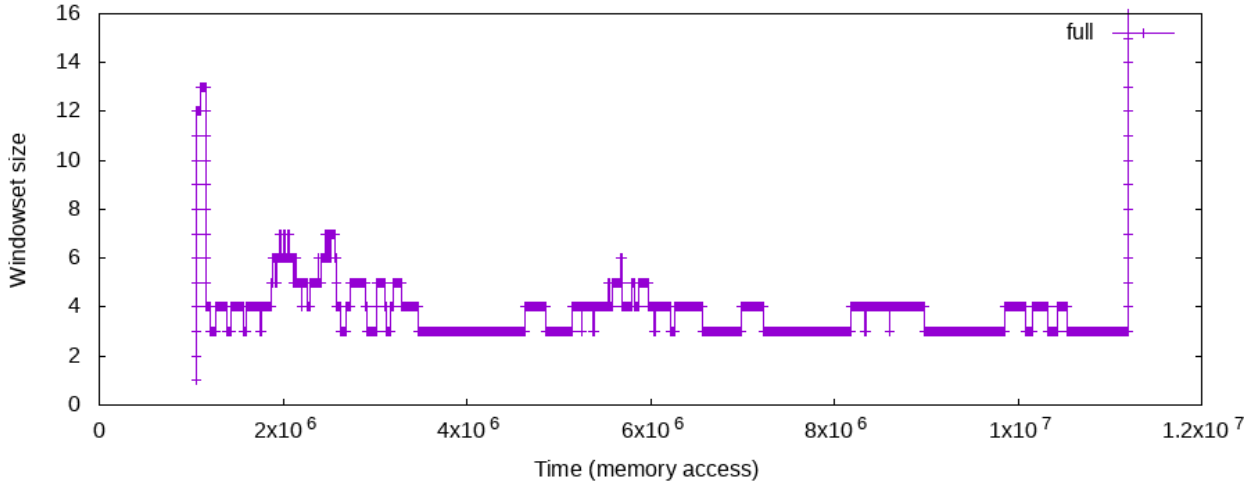

Figure 2.5 Quicksort (data size=10 000; skipsize=1 060 174; page size=8 192; window size=100 000)

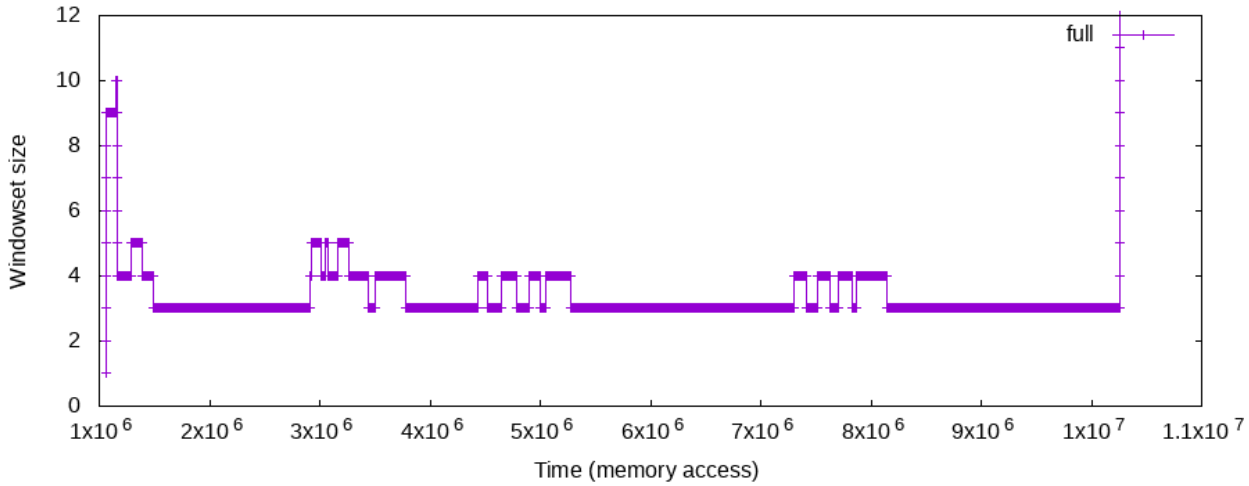

Figure 2.6 Quicksort (data size=10 000; skipsize=1 060 174; page size=16 384; window size=100 000)

My only explanation for the odd spikes would be at points where either a shift in partition happens.

Let's try observing the trend based on modifying the data size.

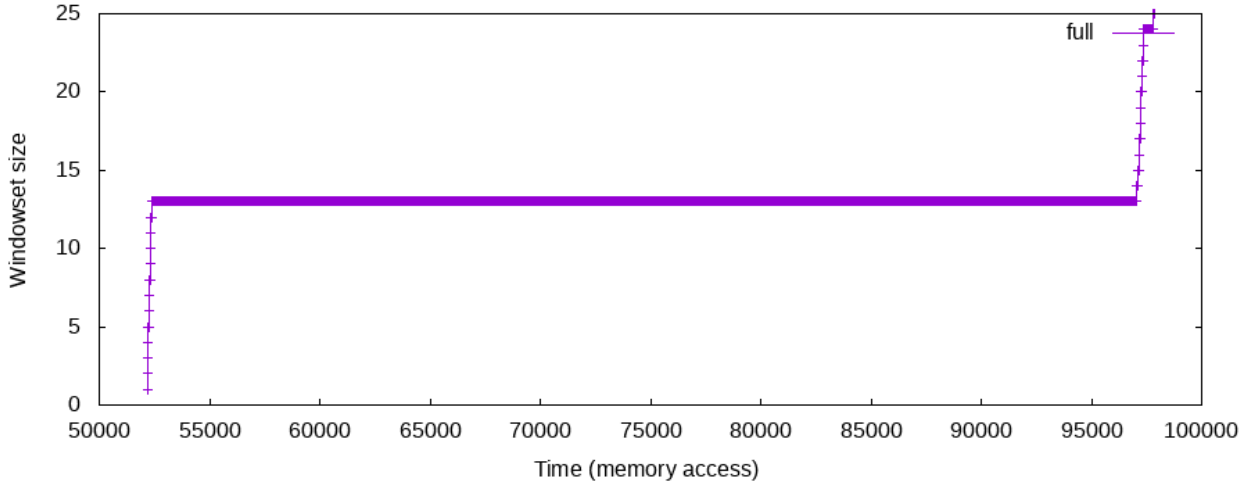

Figure 2.7 Quicksort (data size=100; skipsize=1 060 174; page size=4 096; window size=100 000)

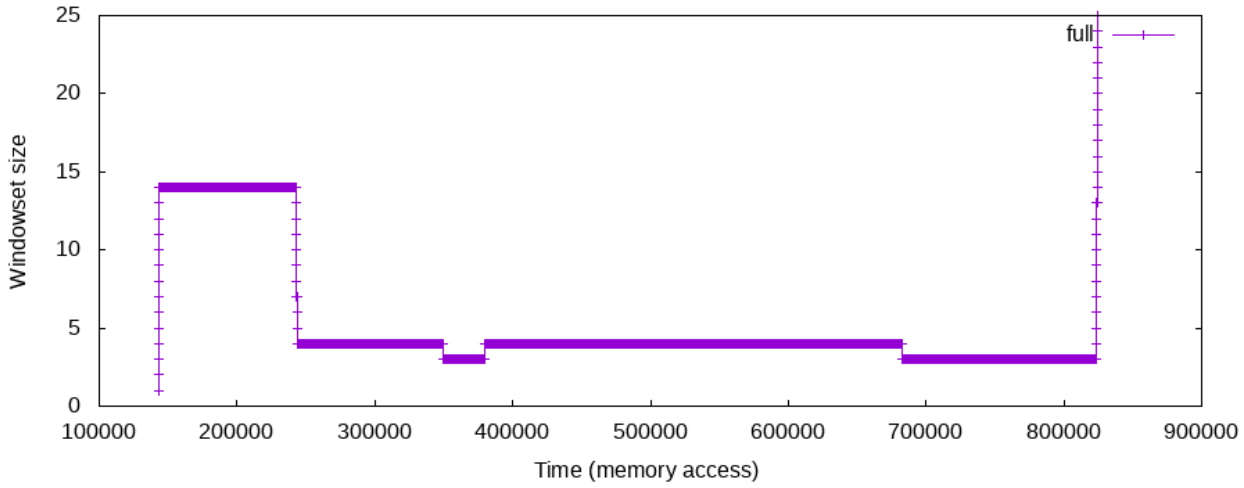

Figure 2.8 Quicksort (data size=1 000; skipsize=1 060 174; page size=4 096; window size=100 000)

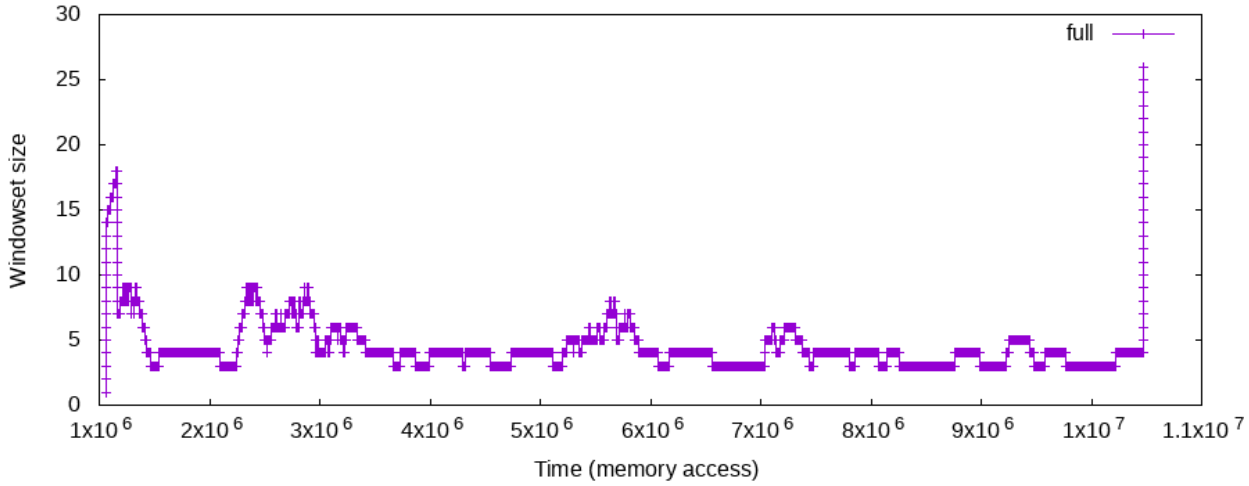

Figure 2.2 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=100 000)

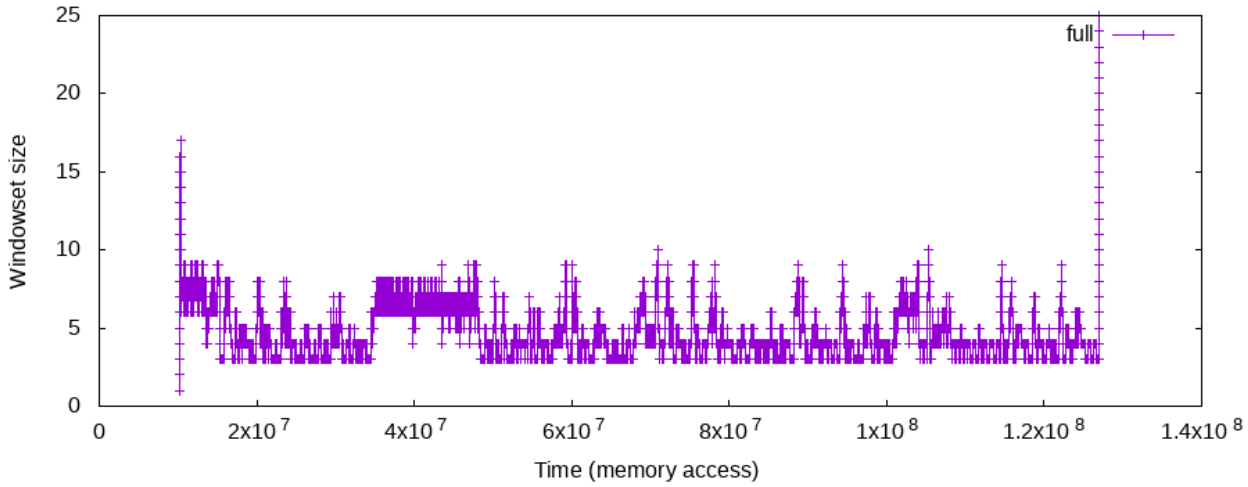

Figure 2.9 Quicksort (data size=100 000; skipsize=1 060 174; page size=4 096; window size=100 000)

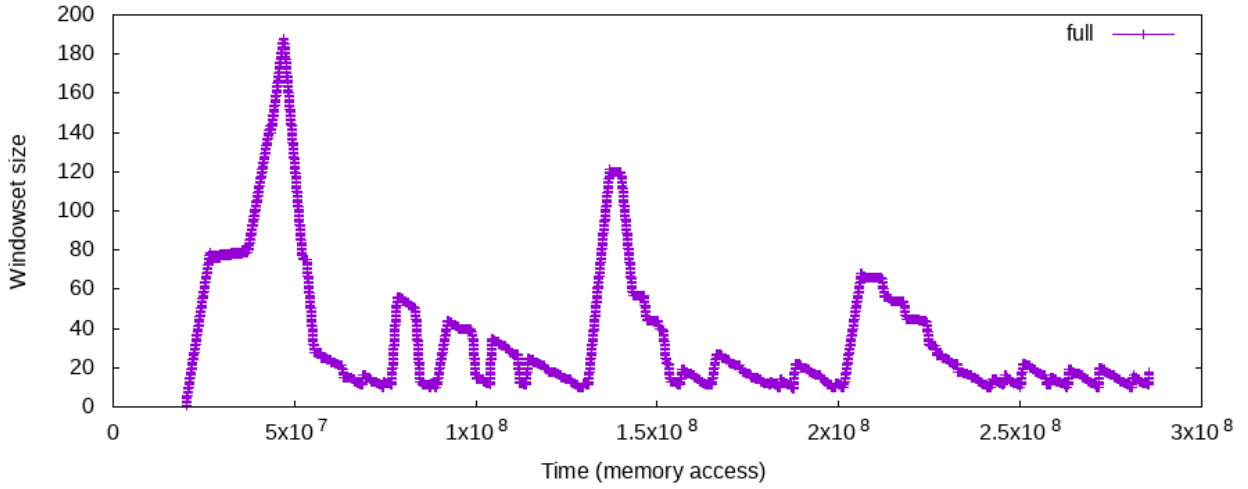

Figure 2.10 Quicksort (data size=200 000; skipsize=1 060 174; page size=4 096; window size=100 000)

The peaks become more apparent when we increased the data size. The peaks may have occurred when we shift in partitions to quicksort, and the previous partitions go out of scope in windowset.

Next we'll observe quicksort's windowset by increasing the window size.

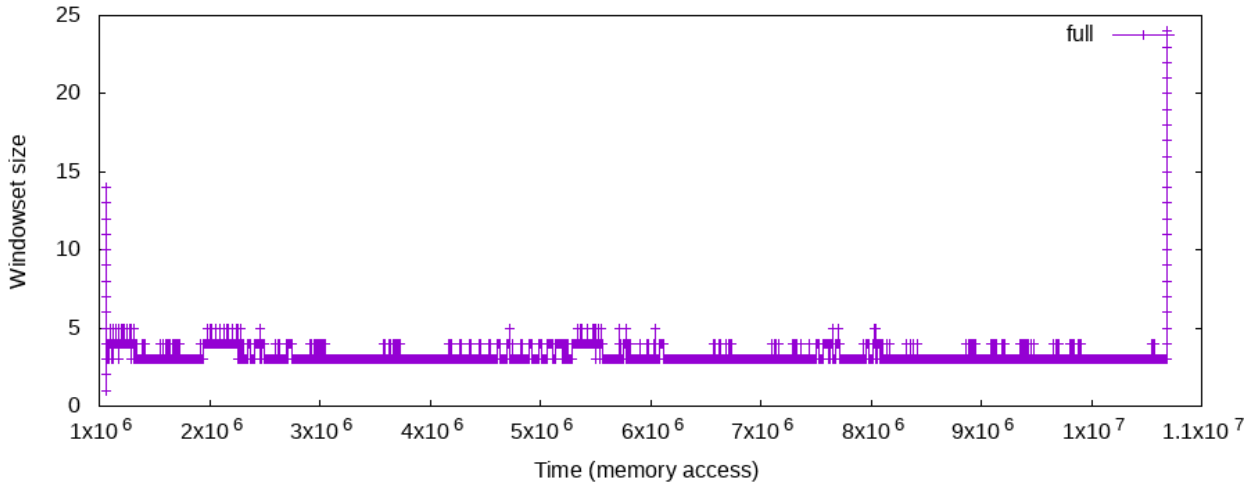

Figure 2.11 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=1 000)

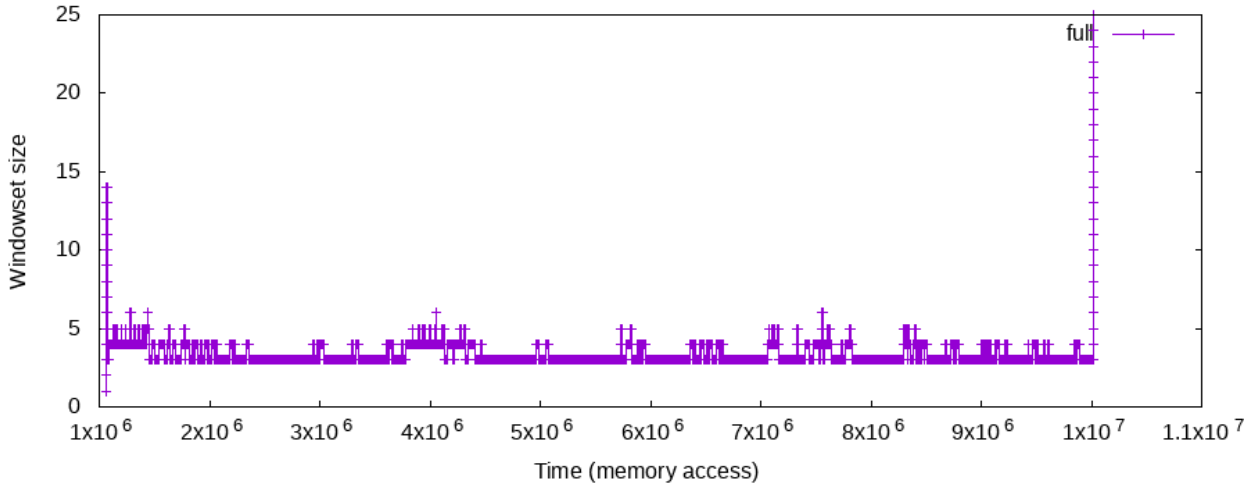

Figure 2.12 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=10 000)

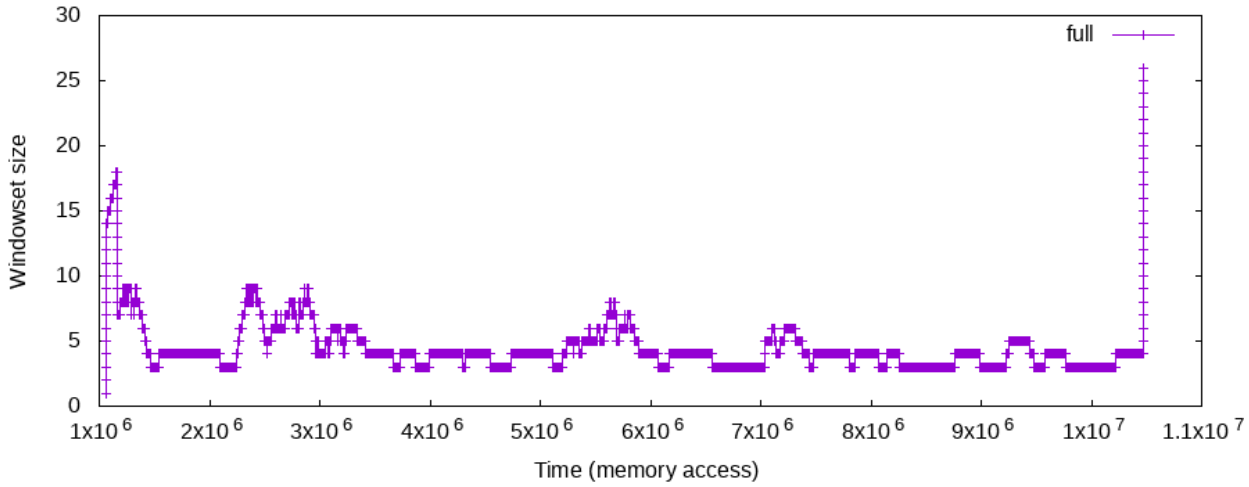

Figure 2.2 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=100 000)

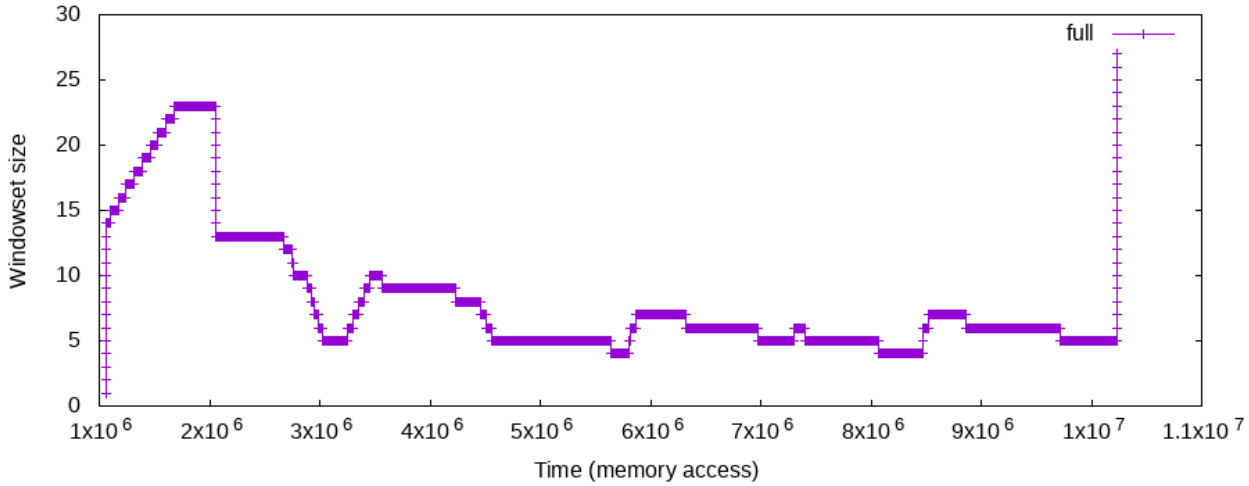

Figure 2.13 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=1 000 000)

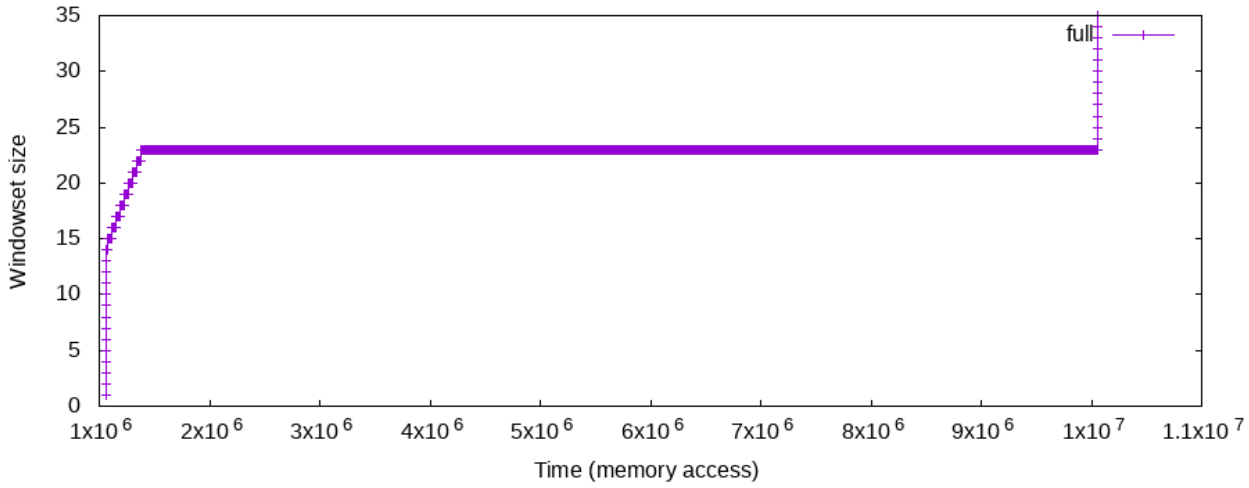

Figure 2.14 Quicksort (data size=10 000; skipsize=1 060 174; page size=4 096; window size=10 000 000)

Increasing the window size too much does not really help find patterns. It becomes a lot more flat, probably indicating there's a steady flow of data going in and out of the windowset, which I think is natural since we go from one partition to another.

3 Radixsort

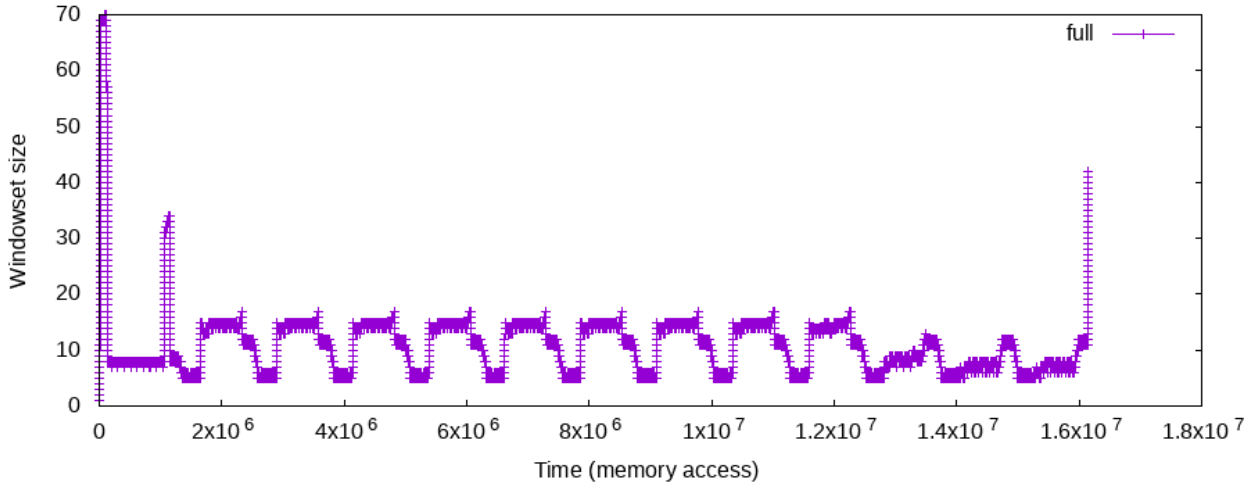

Figure 3.1 Radixsort (data size=10 000; skipsize=0; page size=4 096; window size=100 000)

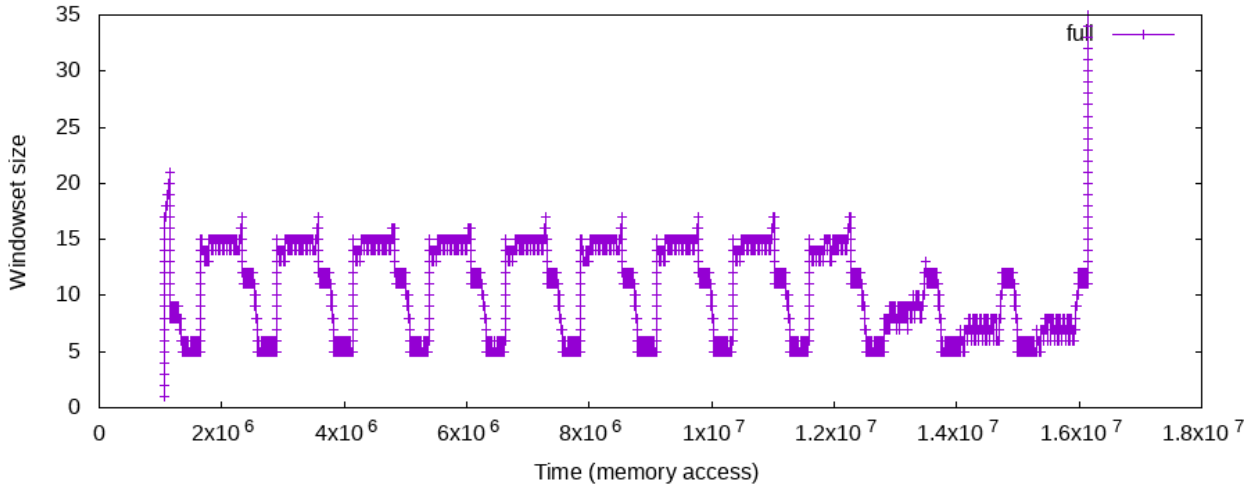

Figure 3.2 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=100 000)

For radix sort, we seem to constantly sort items into positions that are non-contiguous with each other. Numbers that may be near in the least significant digits may end up far from each other when taking into consideration their most significant digits.

Let's observe radixsort in increasing page sizes.

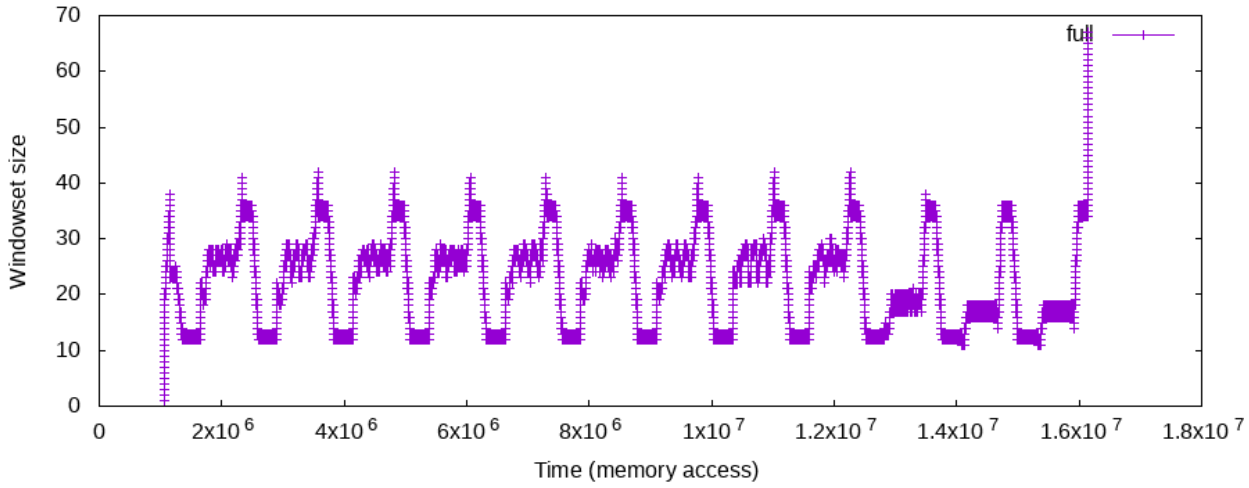

Figure 3.3 Radixsort (data size=10 000; skipsize=1 061 098; page size=1 024; window size=100 000)

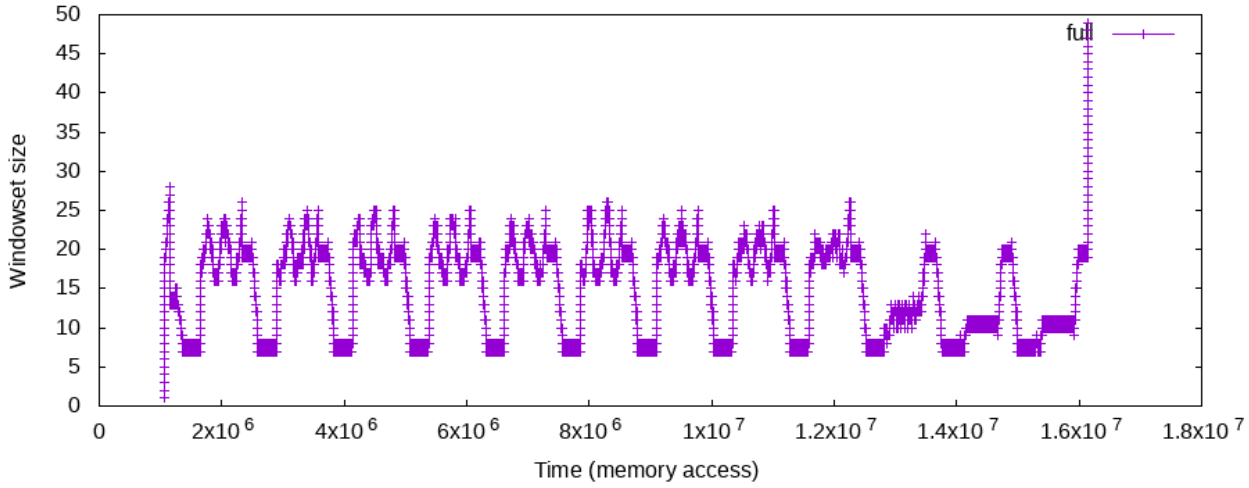

Figure 3.4 Radixsort (data size=10 000; skipsize=1 061 098; page size=2 048; window size=100 000)

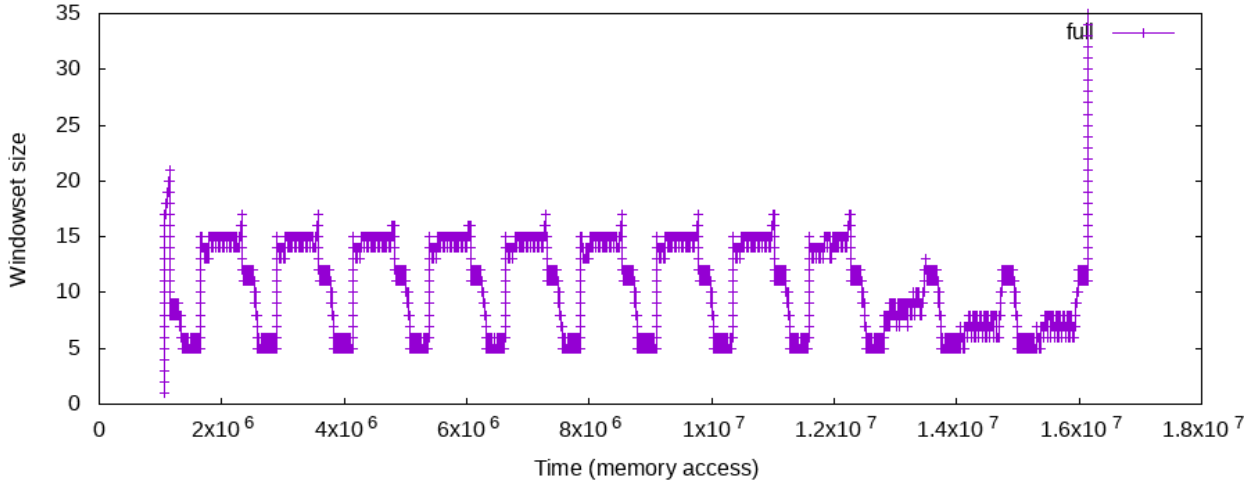

Figure 3.2 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=100 000)

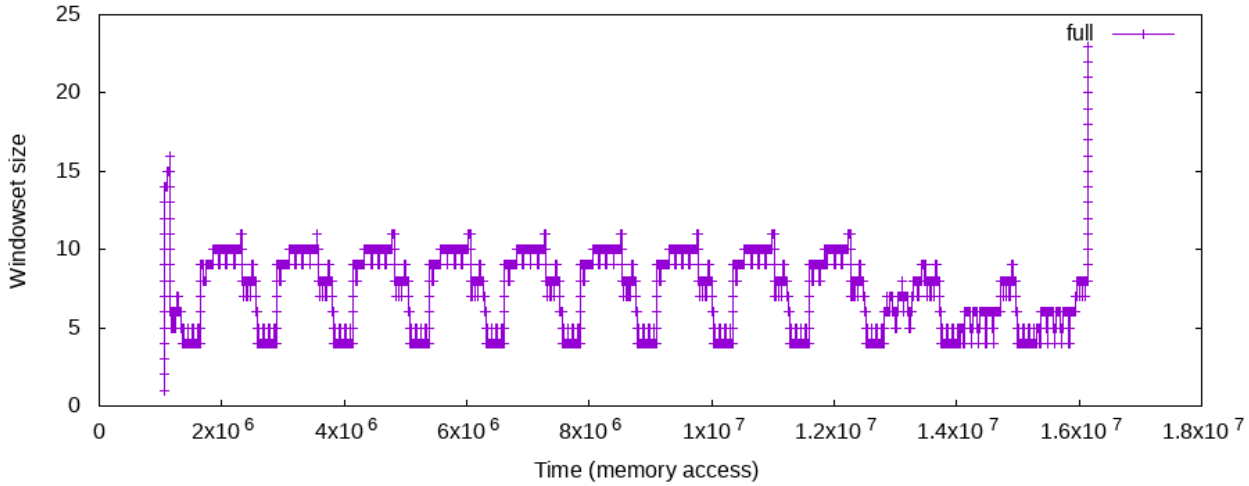

Figure 3.5 Radixsort (data size=10 000; skipsize=1 061 098; page size=8 192; window size=100 000)

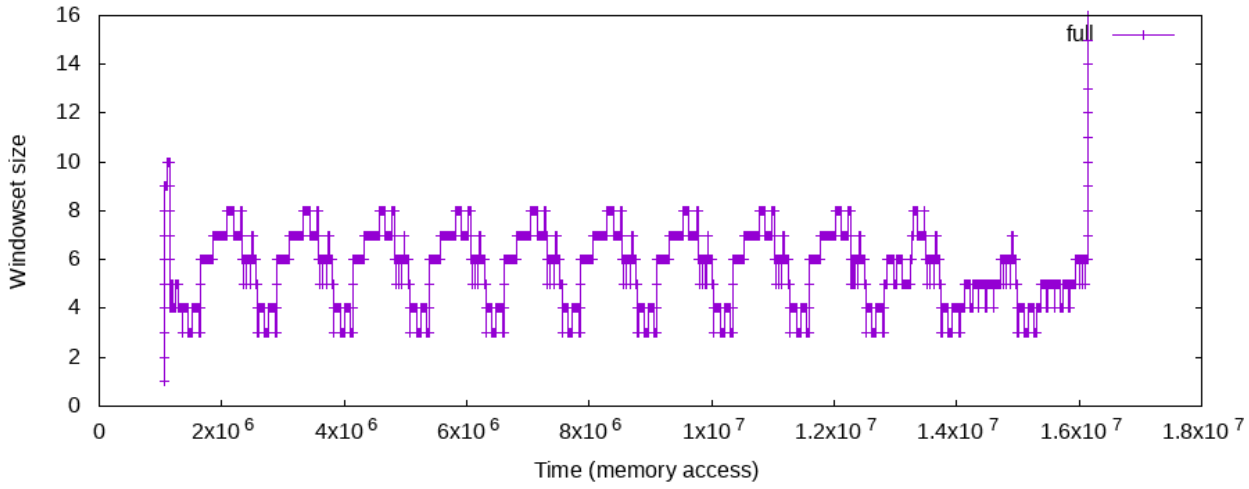

Figure 3.6 Radixsort (data size=10 000; skipsize=1 061 098; page size=16 384; window size=100 000)

Page size increases don't help the fact that radixsort doesn't really put numbers where they should be during the first rounds of sorting. Data is still too far apart for page numbers to help.

Next we'll observe changing the data size.

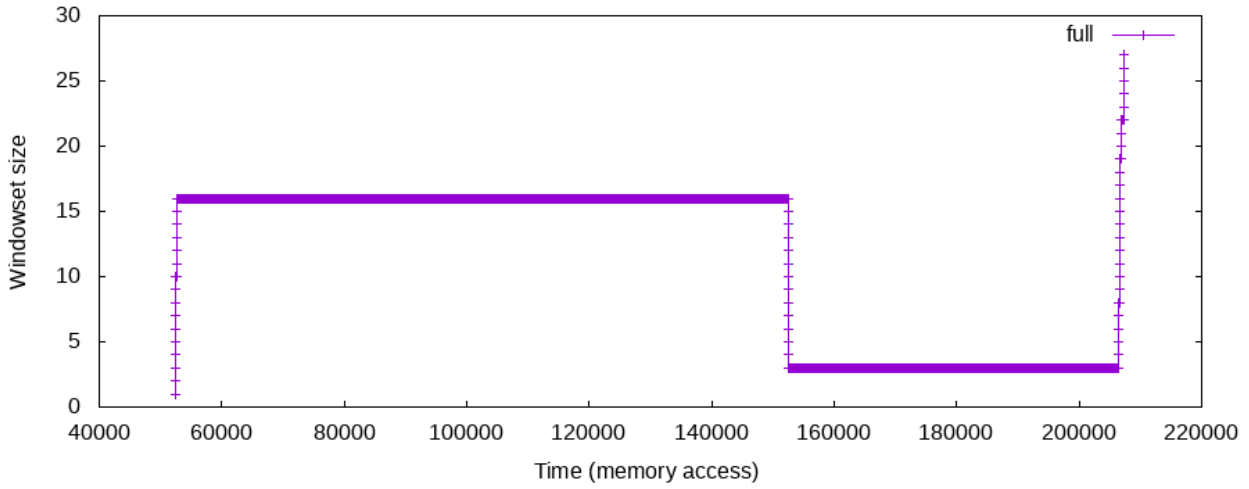

Figure 3.7 Radixsort (data size=100; skipsize=1 061 098; page size=4 096; window size=100 000)

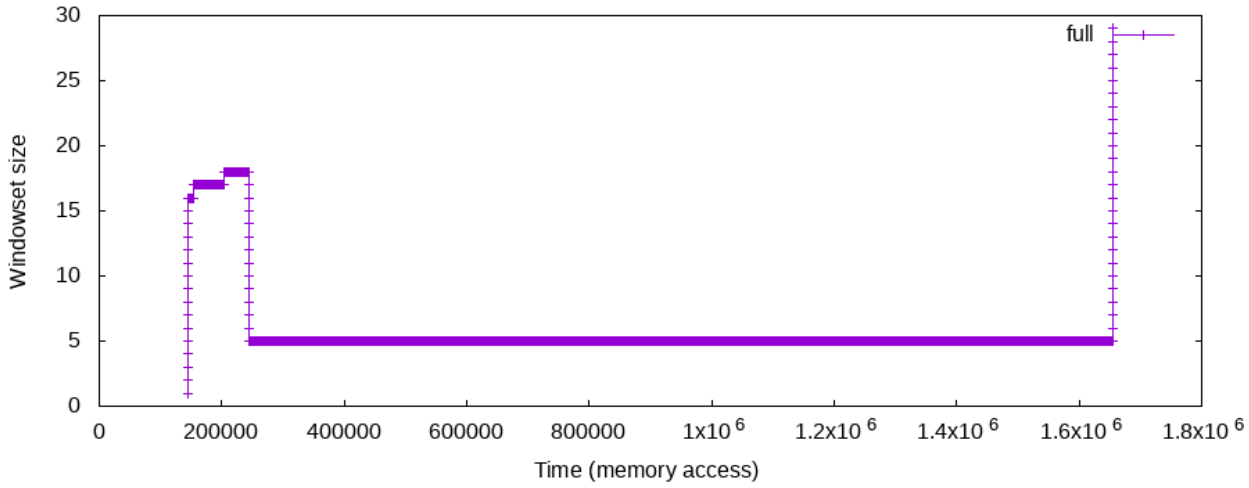

Figure 3.8 Radixsort (data size=1 000; skipsize=1 061 098; page size=4 096; window size=100 000)

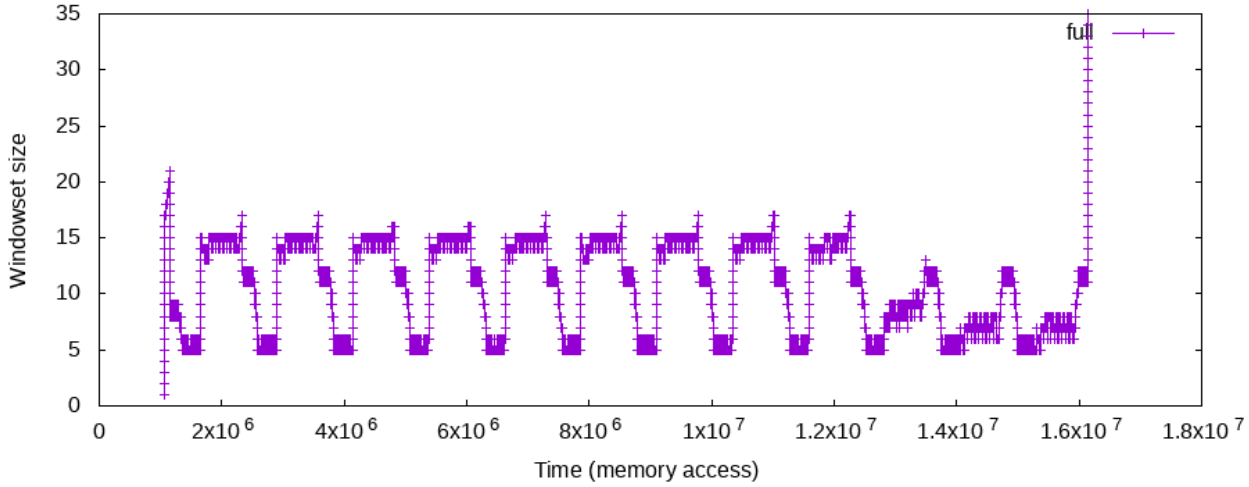

Figure 3.2 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=100 000)

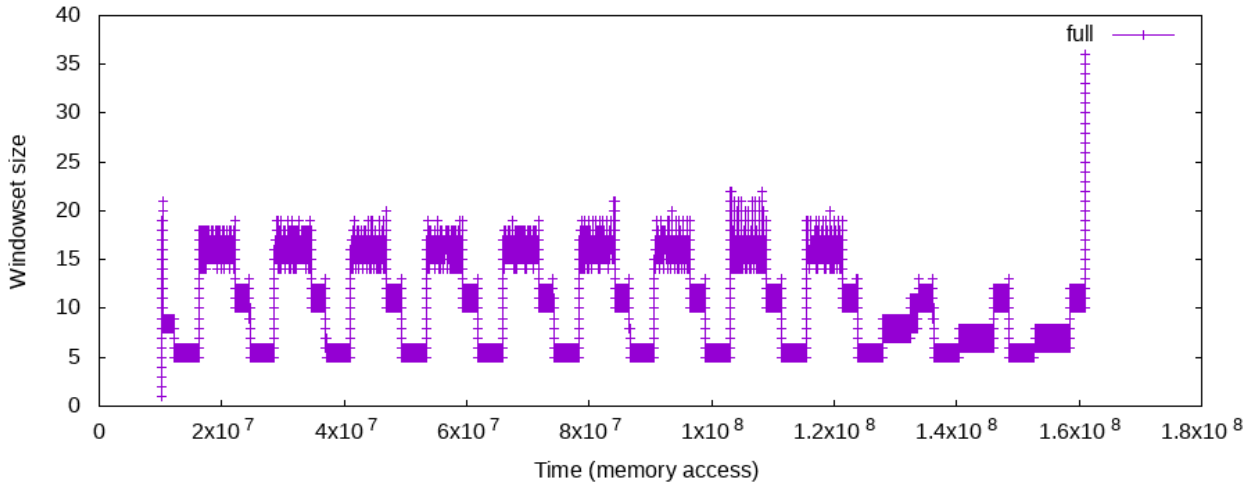

Figure 3.9 Radixsort (data size=100 000; skipsize=1 061 098; page size=4 096; window size=100 000)

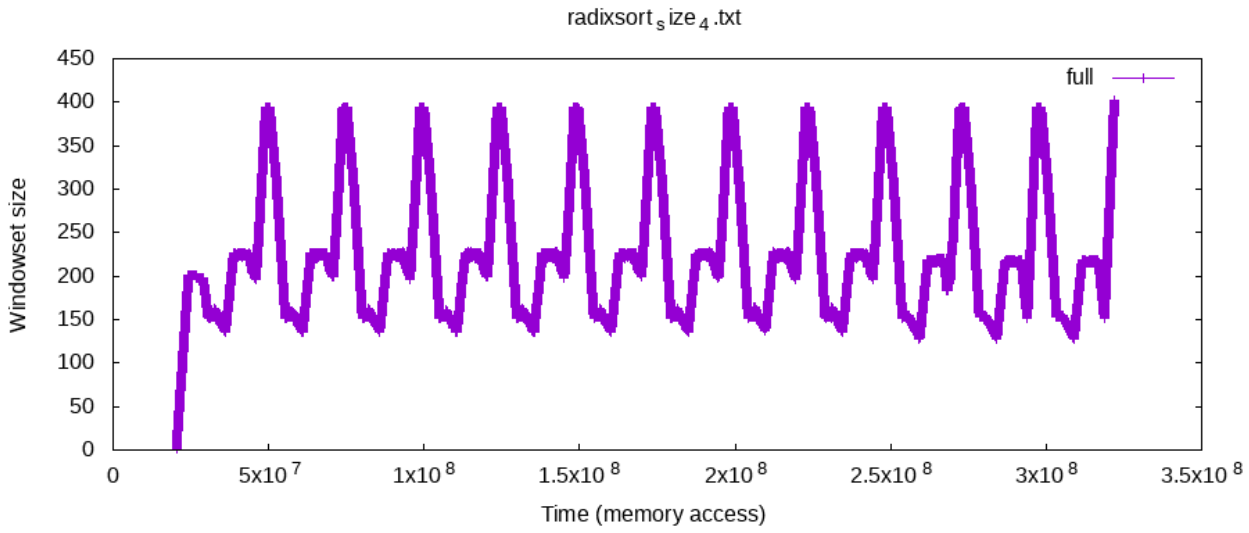

Figure 3.10 Radixsort (data size=200 000; skipsize=1 061 098; page size=4 096; window size=100 000)

Increasing the data size makes my initial observation more apparent where data accessed are consistently non-contiguous.

Let's try increasing the window size.

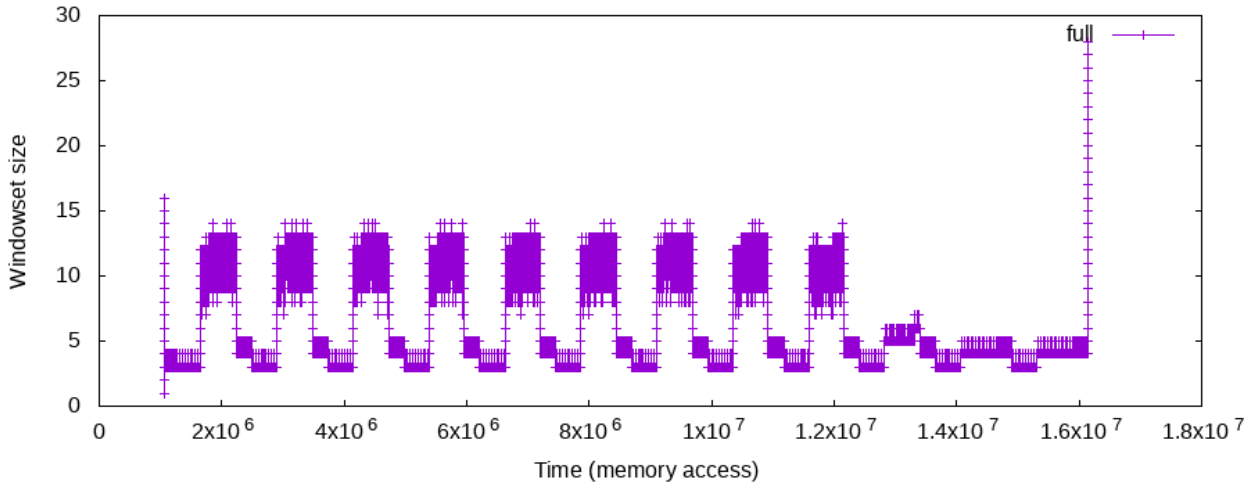

Figure 3.11 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=1 000)

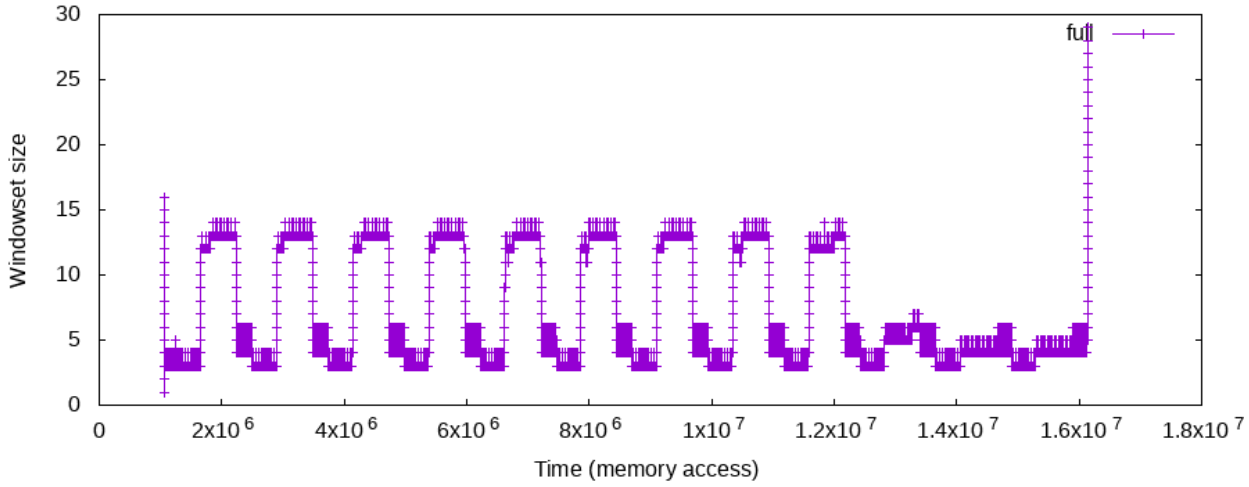

Figure 3.12 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=10 000)

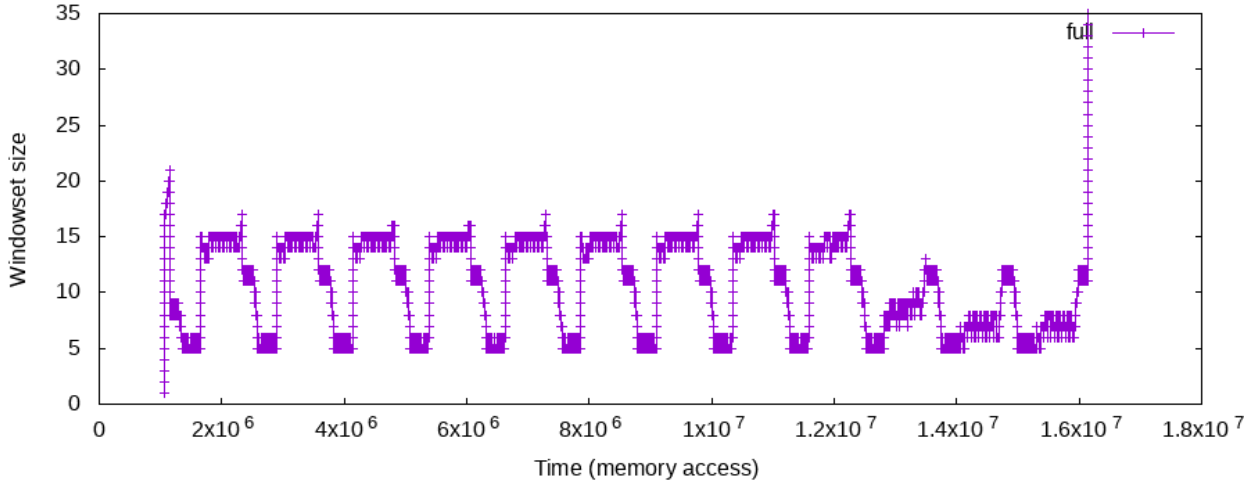

Figure 2.2 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=100 000)

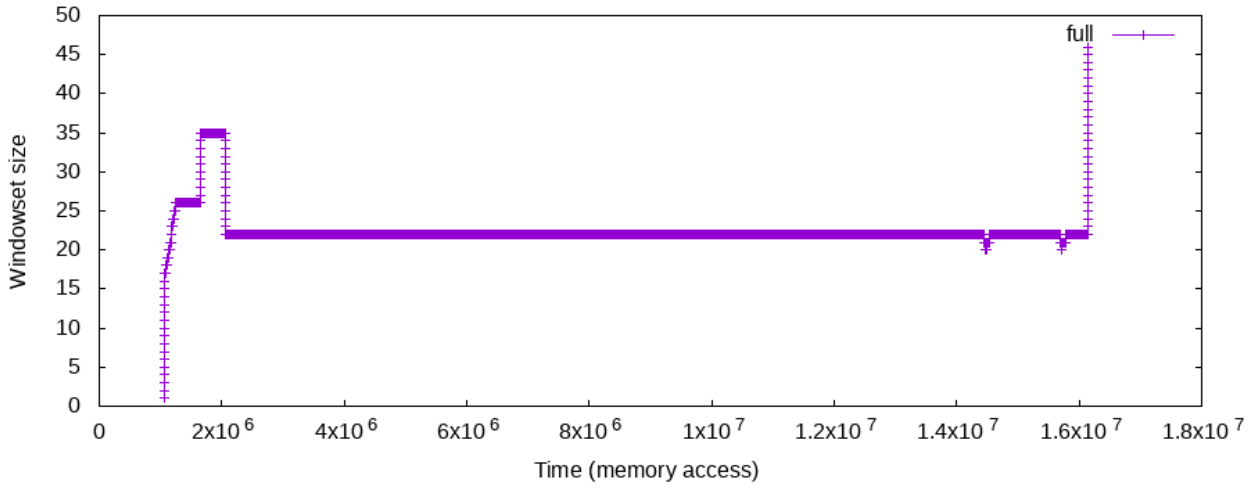

Figure 3.13 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=1 000 000)

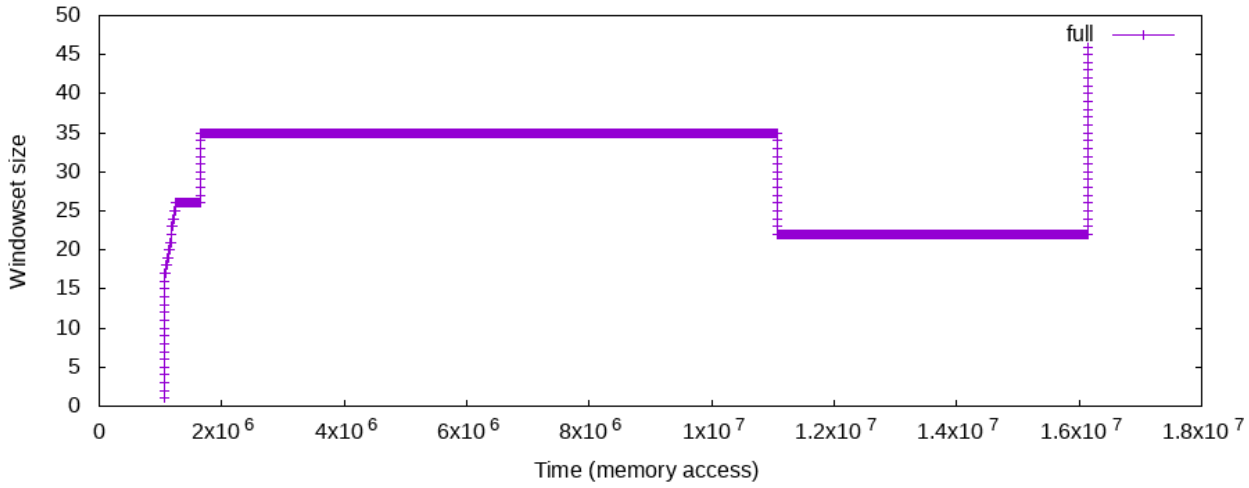

Figure 3.14 Radixsort (data size=10 000; skipsize=1 061 098; page size=4 096; window size=100 000)

Window size increase flattens the consistent spikes in radix sort. Although, radix sort has a bigger windowset size compared to the other sorts, indicating that the memory accesses are still far apart.