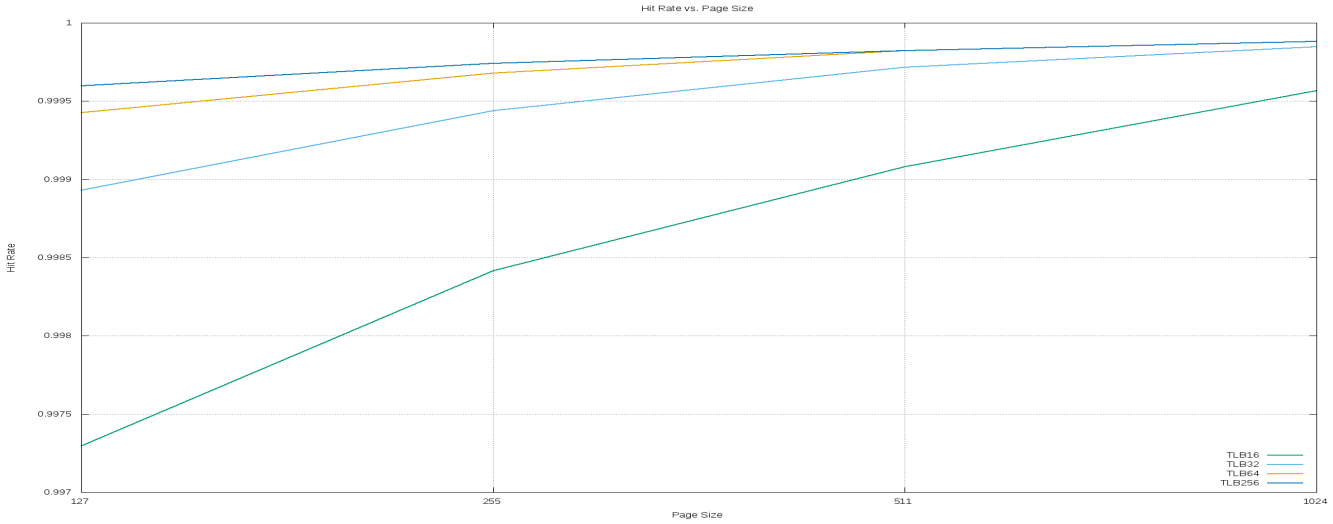
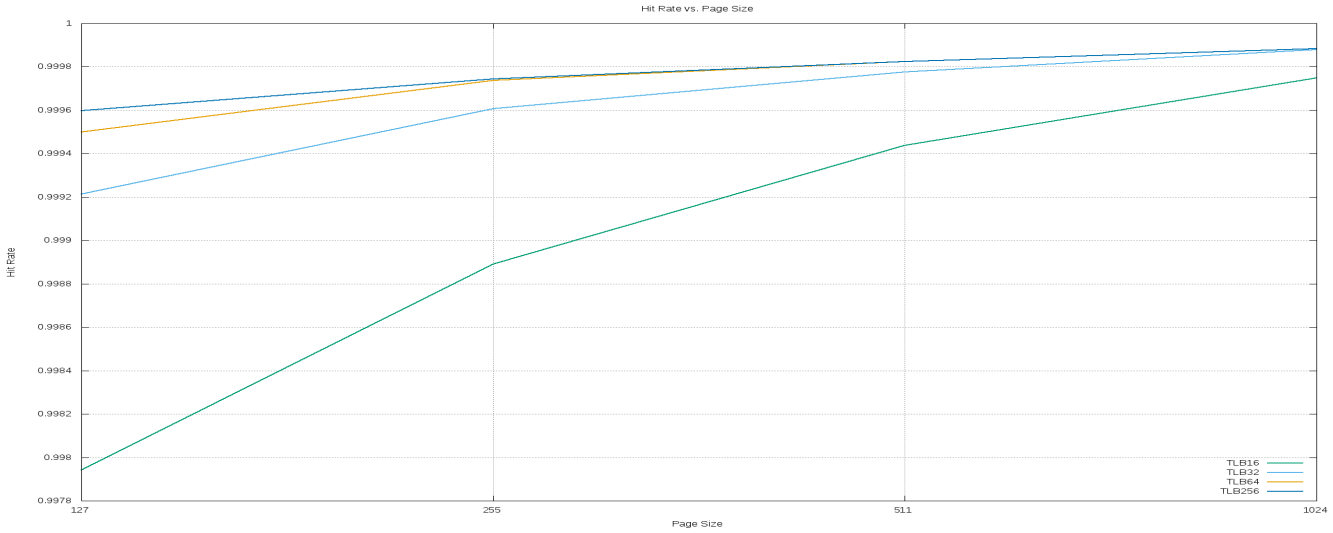
First we compare the performance difference between FIFO and LRU.

FIFO Heapsort 100



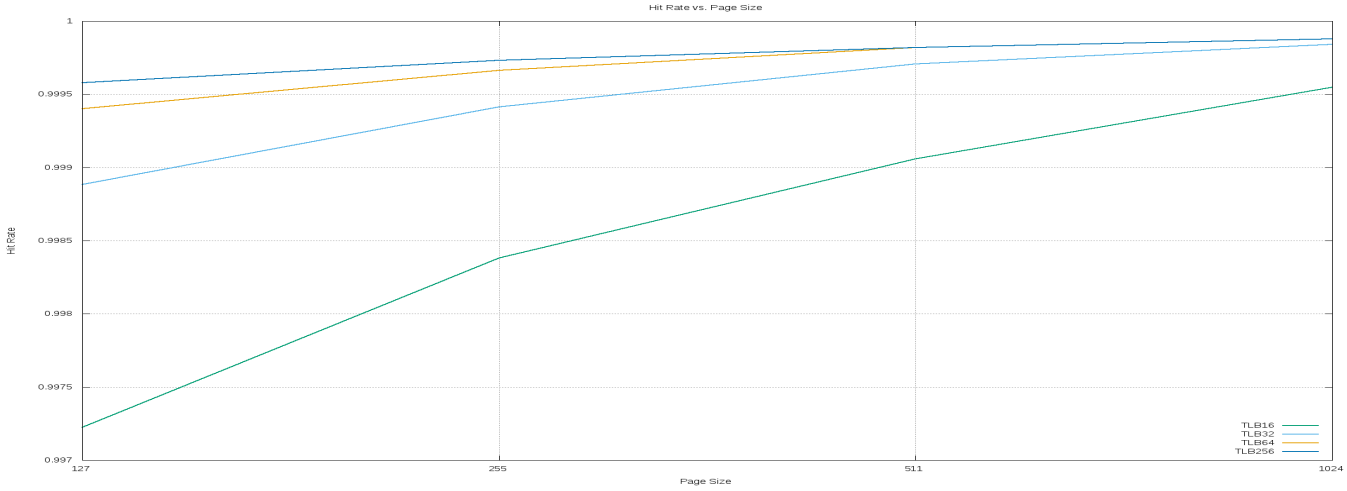
LRU Heapsort 100



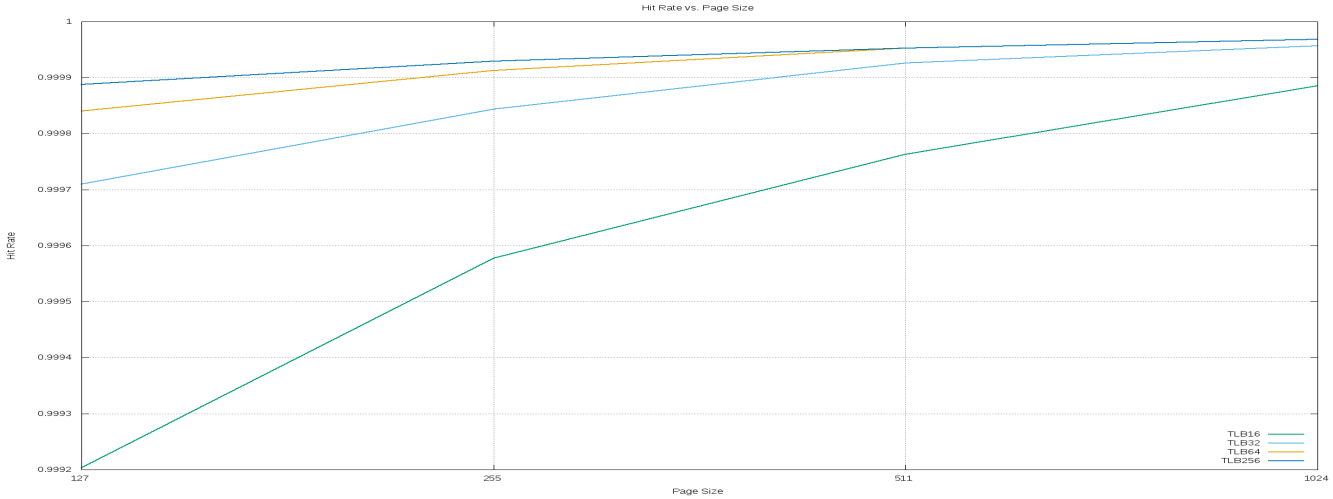
The tlb program using LRU has a higher hit rate over FIFO. This is expected as LRU saves the page sizes that are more recently used compared to FIFO kicking out pages by sequentially. While comparing FIFO vs LRU for other parameters such as bigger inputs and different sorting programs we also get similar results where LRU outperforms FIFO. Since we are doing sorting algorithms the difference between FIFO and LRU are more apparent because these sorting algorithms require the same address space of the numbers being sorted to be accessed many times over.

Comparing how input size effects the performance of our tlb

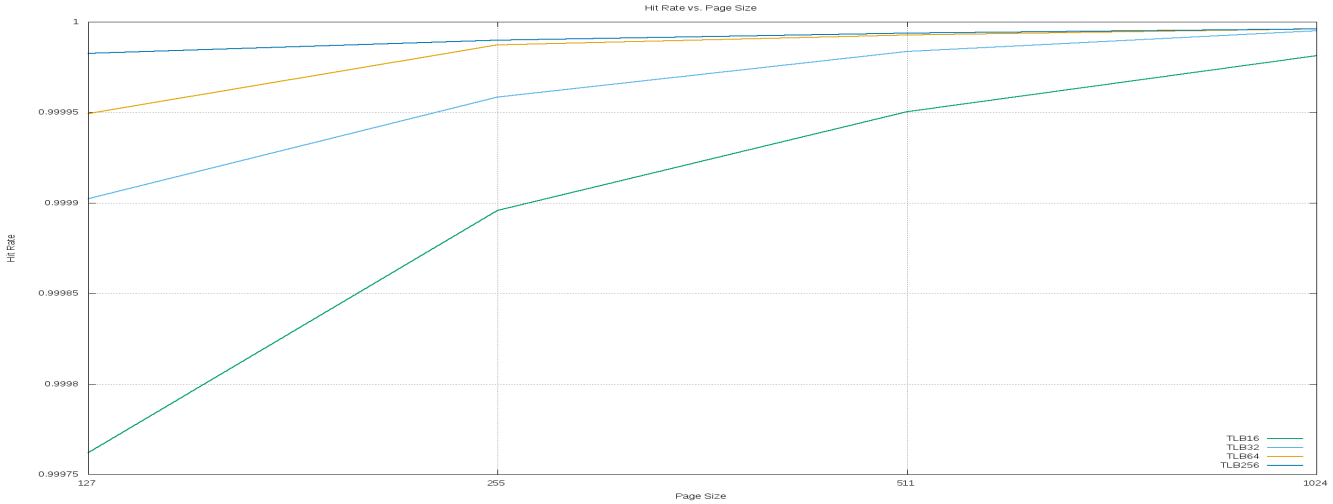
Mergsort 100 FIFO



Mergsort 1000 FIFO



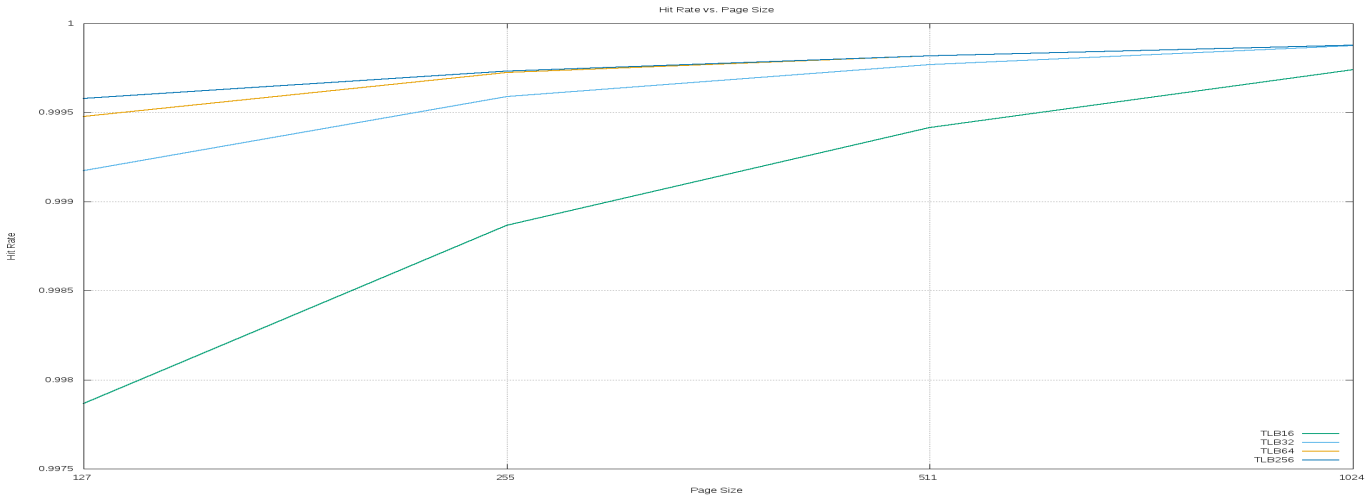
Mergsort 10000 FIFO



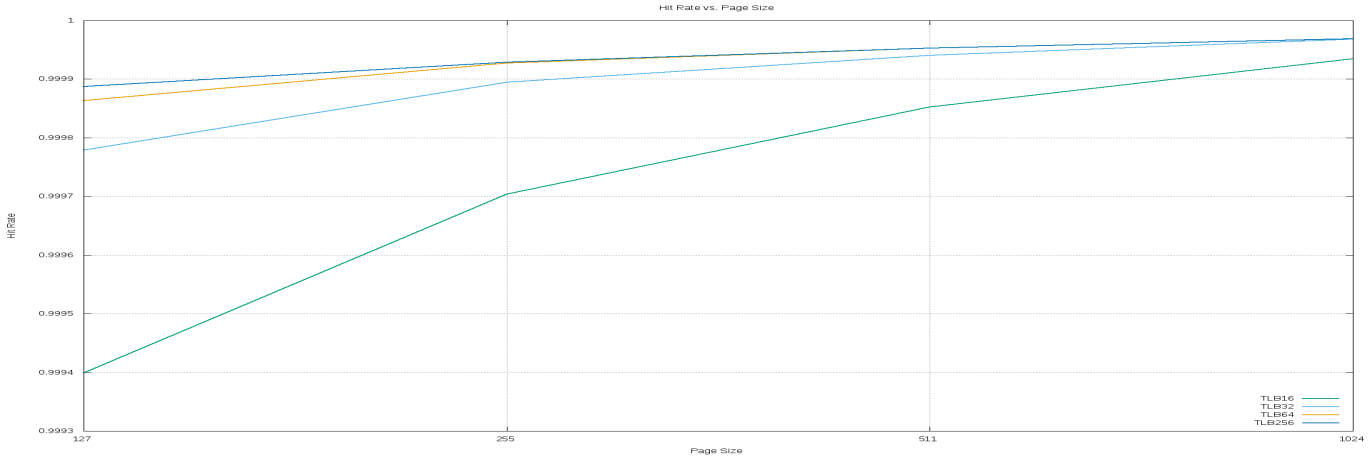
It seems that when we increase the input size of our sorting algorithm. Our tlb actually has a higher hit rate compared to lower input sizes when looking at the same tlb and page sizes. The explanation for this is that initially when we have nothing in the tlb, the miss rate is high. But when the algorithm is finished initializing everything, the algorithm starts sorting the numbers where the miss rate is much lower because we already have the memory locations of the values being sorted (or at least most of it) loaded in the tlb, and the sorting algorithm accesses the those same memory locations of those values continuously which gives us a higher hit rate. And when we increase the input size, this on average increases the amount of times a value being sorted needs to be accessed which is why our hit rate increases compared to lower inputs. The pattern exists between different sorting algorithms as well.

We used the FIFO graphs to as prove our point, but we could see how this would be even more true if LRU was used since LRU prevents the most recently used pages from being kicked out. Shown below are graphs of the same parameters but with LRU instead of FIFO

MERGESORT 100 LRU

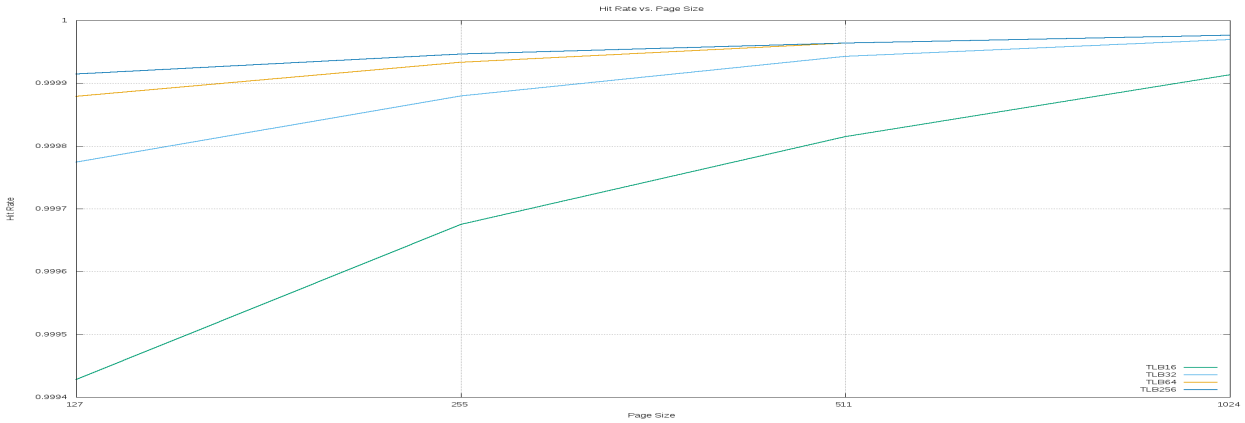


MERGESORT 1000 LRU

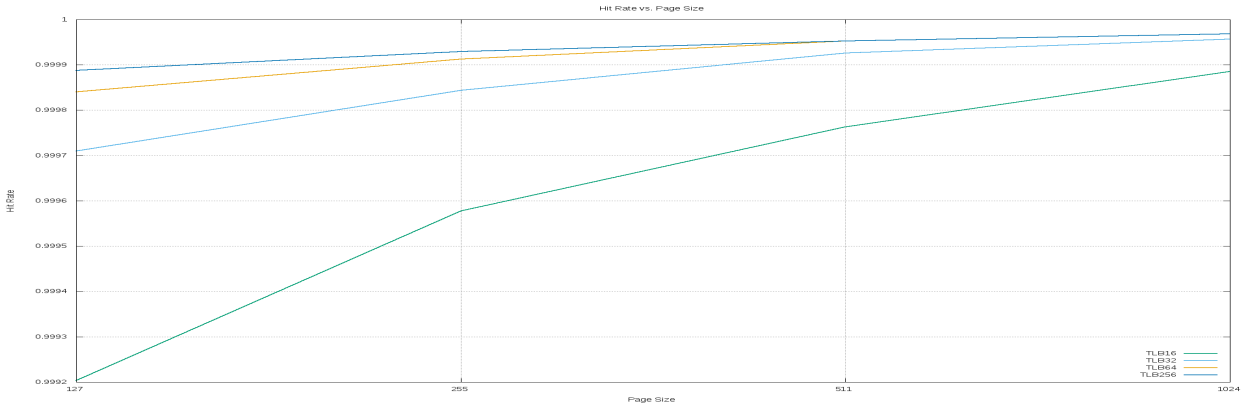


Comparing the effects of different sorting algorithms

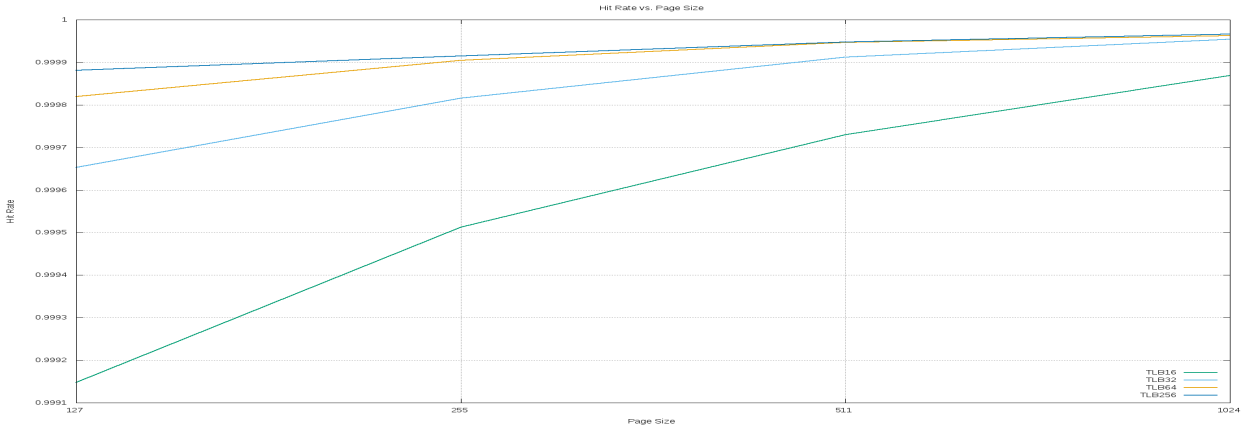
FIFO HEAPSORT 1000



FIFO MERGESORT 1000



FIFO QUICKSORT 1000



When we compare the three sorting algorithms with the same other parameters amongst each other there is little difference performance where Heapsort has a higher hit rate than Merge-sort and merge-sort has a higher hit rate than quicksort. Merge-sort has a higher hit rate than quicksort because Merge-sort uses less recursive calls than quicksort. Merge-sort is guaranteed to have log2(n) layers of recursive calls because it perfectly divides the array into two and sorts them every time, while quicksort sometimes divides the array unevenly which can cause more recursive calls needed to be made. Since each recursive function call enters a new function state, which requires a new stack for saving the register values and hence more memory at another location, more recursive calls generally cause more miss rates. If we look at heapsort, it first calls heapify once, and then calls sort once to sort the numbers, requiring much less function calls than the other two sorting methods which is why it has the best tlb hit rate of the three algorithms.

General findings

Overall we notice from our graphs that larger tlb runs have a higher hit rate than lower tlbs runs. It holds true even when comparing different tlb sizes with different tlb algorithms, different sorting algorithms, or different input sizes. This makes sense since a larger tlb will generally results in fewer misses since we are covering a larger area of the memory.

As we increase the page size with other parameters staying the same, the hit rate increases as well since more memory per page is being loaded into the tlb and so we are more likely to have memory accesses hits compared to smaller page sizes.

In general having more tlb pages seems to increase the hit rate more than having a larger page size. We see this by comparing any two points in the graph where the first point represents a run that has twice the tlb size and half the page size as the second run (for example comparing a run with tlb size 32 and page size 256 with a run with tlb size 16 and page size 512) . The run with the smaller page size but larger tlb will have a higher hit rate than the other, this is because having a larger tlb size represents the memory access locations needed more accurately since we are loading the actual memory locations that the OS is using. While having a larger page size just means if future memory accesses needs memory close to the one we just used then we get an additional hit rate but it is not guaranteed.