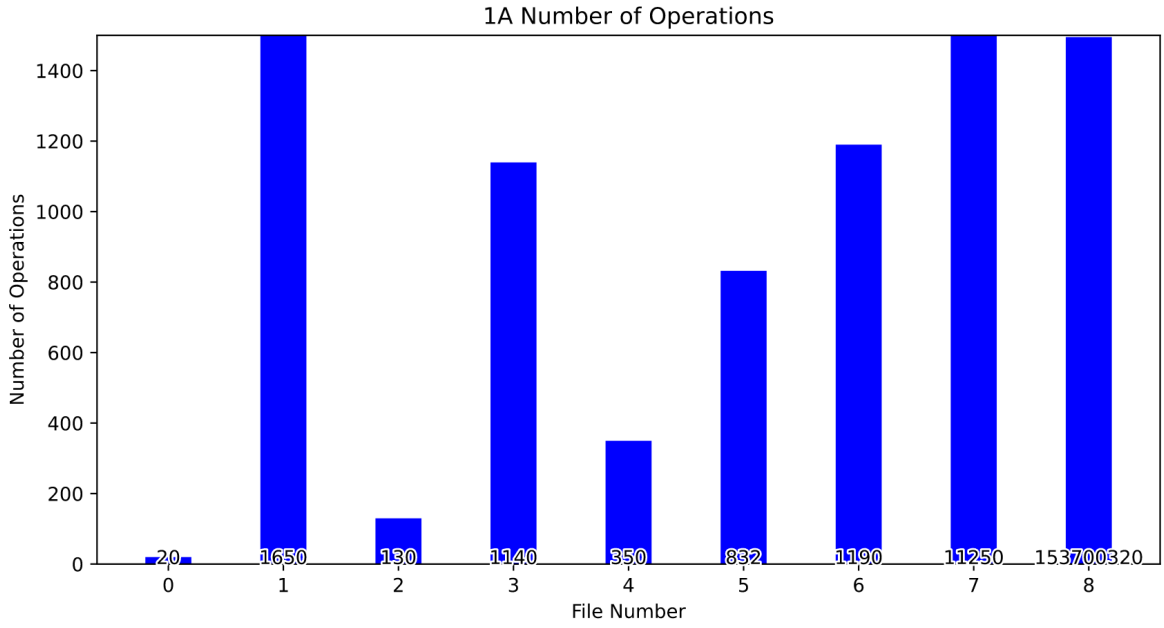
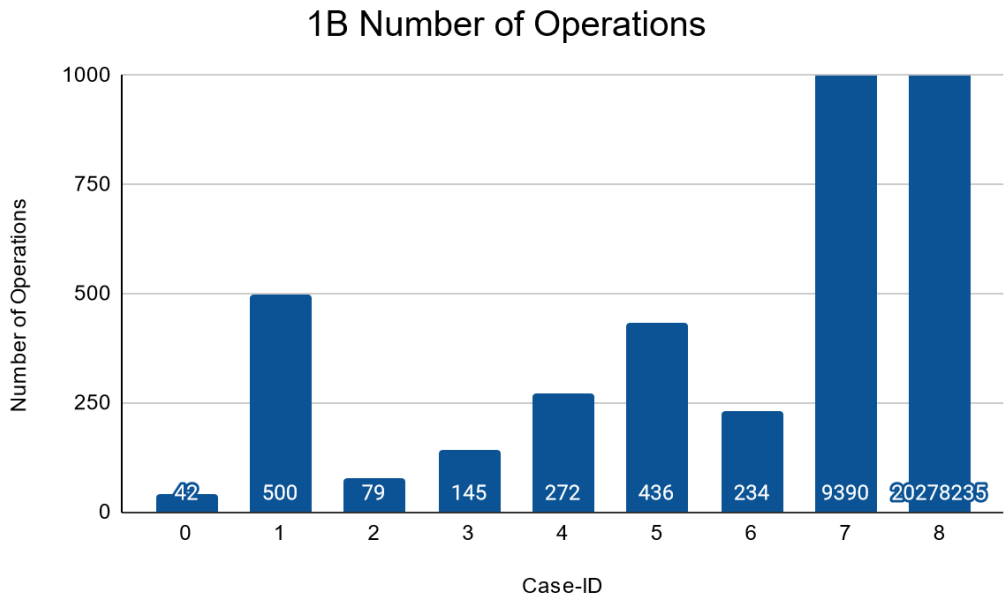
Kendall Fischer and Evan Hatton
CS415 Final Project

Task 1a vs Task 1b:



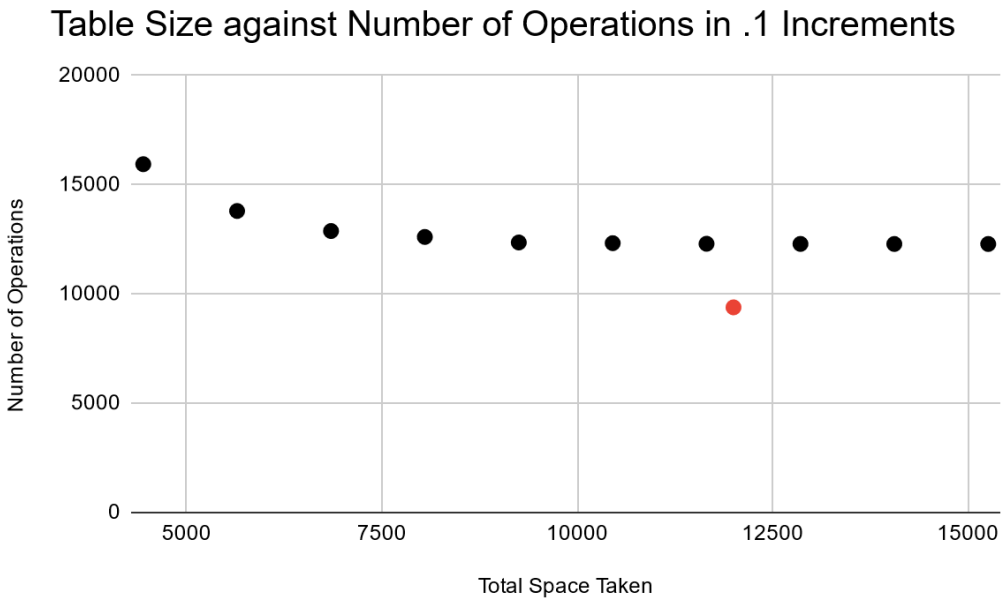


We can see that the 1A naive case takes orders of magnitude more operations than the memory function. Take a look at file #1 for example, because every single cell of the relation has to be computed for 1A, the function takes many extra steps. In the case of the memory function those unnecessary computations are simply omitted. This resulted

in a savings of over 1500 basic operations for file #1, not to mention the differences for file #8.

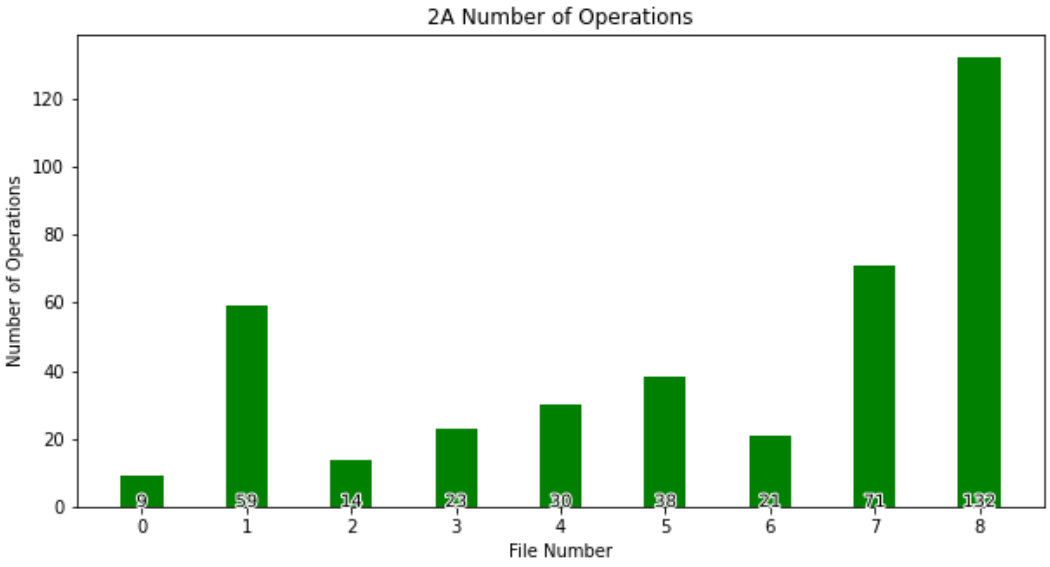
Task 1b vs Task 1c:

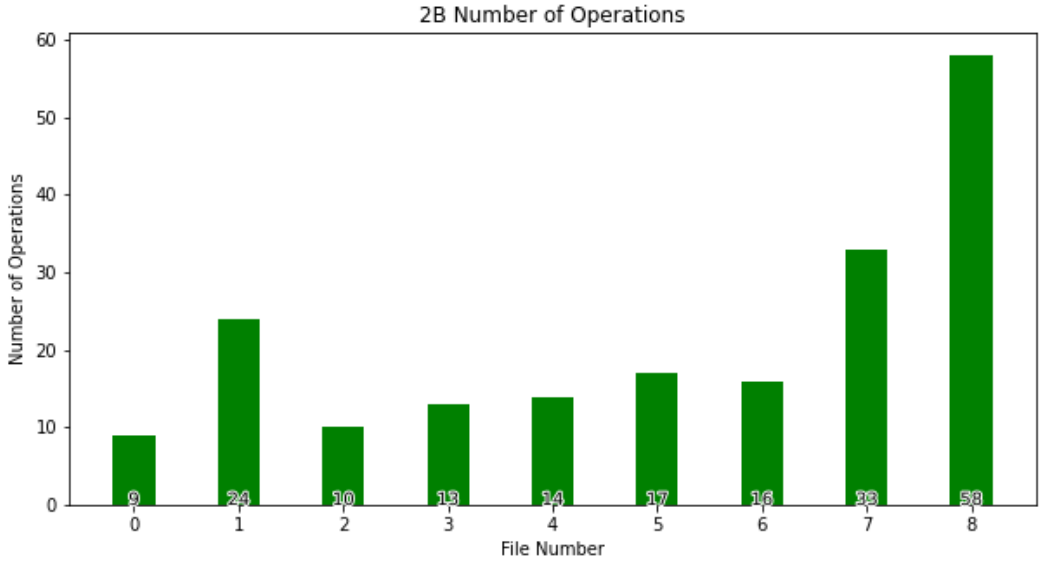
For the memory function approach we assume that the space taken is about half of that of the traditional approach. The hash function should map each position of the table to its own spot in the hash table, and so the the value of k should be greater than or equal to half of the number of rows times columns.



We multiplied the value of n*W by 0.1 increments to get a range of k values. In black is the space taken against the number of operations for the hash table approach. In red is the regular memory function approach, and it is a single point as it doesn't change. The number of operations decreases until n*W is at about half size. Any further size increases for k results in diminishing returns for the number of operations. We used the 7th data set to get these results, as the 8th ran too slowly to repeatedly test. From these results the best ratio to multiply n*W would be either 0.5 or 0.6.

Task 2:





The greedy implementation which uses heaps (2B) is much more efficient than the one utilizing quick sort. This is because the bottom-up approach for initially building the first heap only takes θ(n) operations and then after that *deletemax* requires only θ(log n) operations for each time it's called – so total operations in 2B are n + k log n where k is the number of *deletemax* invocations which at most is equal to n. With the quicksort, the initial sorting operation requires n log n + k operations. Since n + k log n is only equal to n log n + k if k = n, this means that at the worst case 2B will perform at the same efficiency as 2A while it will on average perform better.