Thank you Joel. In addition to Key comparisons, our group also looked at the CPU Time performance of both algorithms.

Using the optimal empirical S value 27 we obtained, we ran the algorithms on different input types and sizes and stored the average CPU Time. On an array of size 100, the HybridSort algorithm outperformed MergeSort on both Random and Sorted arrays, as highlighted in green. However, it did poorer on reversely sorted inputs, highlighted in red. This result is further reinforced as the same trend was maintained on array sized 1000, …. 10,000 …. and 100,000 as well. We compiled the data gathered and expressed the percentage change in CPU Time from MergeSort to HybridSort. As mentioned earlier, on reversely sorted array, the insertion sort portion performs poorest as every element has to be swapped. This would occur at time complexity of n^2. As such any arrays in the recursive function below size 27 would run on the worst case leading to the exponential growth in CPU time taken. For Random and Sorted inputs, insertion sort outperforms mergesort which is why there is an improvement in CPU time performance.

Consolidating, our results, you might wonder why despite a higher number of comparisons on random arrays, there was a shorter time taken for HybridSort still. By looking at the average complexity of both algorithms, one must also consider the constant multiplier. In this case, the difference in constant value is likely contributing to why HybridSort is faster despite more comparisons. Furthermore, by taking the derivative with respect to n, it is clear that HybridSort time complexity is not only dependent on n but due to other factors as well such as the S value.

To conclude, HybridSort is an algorithm that can achieve a space complexity of O(n) by performing internal sorting during the merge function. Additionally, the Theoretical S value differs from the Empirical S value determined. Our group observed from repeated testing, that the optimal S value seemed to vary over a range of values instead of one. This could be due to the memory usage and performance between computers differing. The 3D plot below shows that as S varies, the time taken is lower across a certain range of values with varying array sizes. Taking into account that merge function performs better on array sizes that are in powers of 2 and our results, our group concluded the optimal range of S could be from 16 to 32. Moving forward, the HybridSort algorithm can also be improved by introducing a Galloping method from TimSort, which is another Hybrid MergeInsertion sort algorithm that achieves best case complexity of O(n).

With that, we have come to the end of our presentation. Thank you!