Good afternoon Prof? and fellow peers, I am Jabez, this is Augustine and Jun yi and we will be presenting on our project regarding hybrid sort which is the integration of merge and insertion sort.

Moving on, this is our scope for the project.

Starting with our hybrid sort algorithm, this firstly involves the use of the merge function from merge sort. In this implementation, we utilized the merge function using an auxiliary array. So there is no in place sorting taking place. This essentially involves iterating through each of the arrays, L and R and inserting the elements into the array result in a sorted order.

The key comparisons are being tracked using an initialized attribute called ‘comparisons'

We also utilized insertion sort for this hybrid algorithm and this is our implementation of it. Now moving to our hybrid algorithm. In essence this algorithm recursively divides the array into 2 sub arrays, namely L and R, before merging the subarrays together in a sorted order. This is the essence of mergesort. However, when the array size reaches a threshold S, it switches over to insertion sort due it functioning more efficiently with smaller sized inputs.

Next we will explain briefly on our data generation process. We generated 12 columns with their sizes relating to the input size. 2 of the columns were data in ascending and descending order while the last 10 were randomly generated.

The input sizes that we generated were of size 1K, 10K, 100K, 1M and 10M. We accounted for the best, average and worst cases which are sorted, random and reverse order. Do note that the worst case refers to the insertion sort component of hybrid sort. For merge sort, all forms of sorted arrays are the best case.

Moving on to the time and space complexity. We will analyse this using a tree example where S = 2. In this case, we can see that we will end up with n/2 subarrays before needing to perform insertion sort on each of these n/2 subarrays. Because we have n/2 leaves, the height of the tree is log(n/2). For each of the tree levels, we have to perform merge operation on the arrays at the level which takes O(n) time. With this information, we can then calculate the time complexity for each of the 3 cases. So as we can see because tree height is n/2 so merge sort T.C is nlog(n/2) for all 3 cases. Because subarrays are <= to size s, thus time complexity fi O(s) for best and O(s^2) for average and worst case. Using this, hybrid sort best case will be nlog(n/s) for the merge component + s(n/s) because we peform insertion sort for the n/s subarrays. And this gives us O(nlog(n/s)+n. Similarly for average and worst case, hybrid sorts time complexity is blah and blah respectively.

Space complexity for hybrid sort is O(n) because of the use of an auxillary array.

We will now analyse the key comparisons against the 5 input sizes, while fixing the S value at 7. And…. Here are our results! The number of key comparisons was also calculated using the theoretically derived time complexity function as mentioned previously.

Here is the plot for the 3 different cases and we can see that there is an exponential increase in key comparisons as data input size increases. We will also show how the empirical results fared against the theoretical results for each of the cases. So here is the plot for our best case, average case and worst case. We can see that generally the trend is similar.