Hi, I am Xuanye. In my part, I will talk about why we use hybrid sort. Previously, shaofeng mentioned that we need to find a sort algorithm more efficient than merge sort.

To trace back, we have learnt two sort algorithms in previous lecture, Insertion sort and merge sort and their complexity. It’s obviously that insertion sort won’t run faster than the merge sort. Because we know that nlogn always less than n square when n>=1. It seems that the merge sort is already the most efficient. But actually, we still can optimize it. The actual time complexity of these two sorts are shown here. And when we draw the these two functions, can find that when array size n<=7, insertion sort runs a little bit faster than merge sort. This could be the point that we do our optimization.

Our idea is that when doing merging until the subarray size is already small where we use insertion sort more efficiently than merge sort, then we stop to merge the subarray and change our sort algorithm to insertion sort. This makes our hybrid these sorts we have learnt. We define the subarray size to S.

Here we write a function to compare insertion sort and merge sort more clearly. When subarray size S ᕮ [3 , 7], using insertion sort is more efficient than merge sort. As the graph shows that when s = 1 or 2, their efficiency are the same. So we determine our hybrid sort rule that whenever the subarray size merge below 8 or <= 7, we use insertion sort instead of hybrid sort.

Due to different input cases every time will make different minimum subarray size either can be 3, 4, 5, 6, 7. Although we find the biggest difference between these two sort is when subarray size is 5, we would like to choose the optimal S to be 7 instead of 5 since in 7 case it also contain subarray size could be 4,5,6,7. But in 5 case, it don’t have 6, 7 case and may do one more merge than 7 case sometime.