Problems

2.1

1. We have n/k sublists => n/k \* O(k^2) = O(nk)
2. Merge the n/k sublists, this will take lg(n/k), and we make n comparisons => in total: O(nlg(n/k))
3. O(nk + nlg(n/k))

k=lgn

=> O(nk + nlg(n/k)) = O(nlgn + nlgn/lgn)= O(nlgn)

1. K has to be the largest input size for which insertion sort runs faster than merge sort.

2.2

1. Check that A` consists of the original elements in A (but in sorted order)
2. Initialization: initially, the subarray contains only the final element A[n] when the execution of the inner loop starts

Maintenance: On each step, replaces A[j] with A[j-1] if it is smaller and as a result A[j] becomes the smallest

Termination: The loop terminates when j=i+1. A[i] is the smallest element and the array has the original elements but now in a different order.

1. Initialization: We have the empty array

Maintenance: After the execution of the inner loop A[i] will be the smallest element in the subarray => at the end A[i]<A[k], i<k

Term.: the loop termites with i=n (n is the length of the arr). Finally, we have the array A[1…n] with its original elements but now in sorted order.

1. Worst case: iterates over the whole array for each element => performs n comparisons and swaps, so O(n^2)

Insertion sort also has a worst case of O(n^2), but the num of swaps in bubble sort is much much more than in insertion sort => the constant C(bubble)>>C(insertion). For the same n number of elements in the array, insertion sort will run much faster.