(2) **(Hybrid sorting)** (30 points) Insertion sort is one of the fastest algorithms in practice for sorting an array of length n when n is small. Its worst-case running time is $\Theta(n^2)$, however, so for large n merge sort will be faster in the worst case as it runs in $\Theta(n \log n)$ time.

Consider the following *hybrid* sorting algorithm, which tries to combine the best features of insertion sort and merge sort. Suppose we divide the sorting problem into subproblems as in merge sort, but use insertion sort to solve a subproblem once it is small enough. More precisely, suppose we divide the input array into $\lceil n/k \rceil$ lists of length at most k, sort each list using insertion sort, and then merge them into one sorted list, where k is a parameter.

The following questions analyze the running time of this hybrid algorithm.

(a) (10 points) Show that $\lceil n/k \rceil$ lists, each of length at most k, can be sorted by insertion sort in $\Theta(nk)$ worst-case time.

Since we will have $\lceil n/k \rceil$ lists each of length k, running insertion sort on a list will have a worst case time complexity of its length squared which would be $\Theta(k^2)$.

Then the number of groups we have to sort will $\lceil n/k \rceil$

So the total time will be the number of calls to insertion sort times the time it takes to sort them

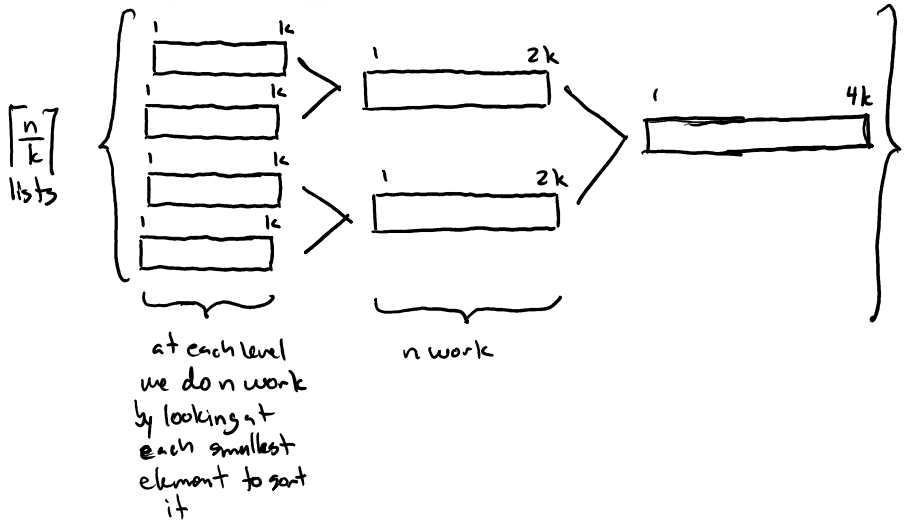
$$\lceil \tfrac{n}{k} \rceil \cdot \Theta(k^2)$$

$\lceil \frac{n}{k} \rceil$ → calls to insertion sort

$\Theta(k^2)$ → time for insertion sort to run on a list of length k

$$= \Theta(n/k) \cdot \Theta(k^2) = \Theta(\tfrac{n}{k} \cdot k^2) = \Theta(nk)$$

(b) (10 points) Show that the sorted sublists from Part (a) can be merged into one sorted list in $\Theta(n \log(n/k))$ worst-case time.



We will have $\log_2(\lceil n/k \rceil)$ levels because we only split until we have lists of length k.

To get the total time we multiply the number of levels times the amount of work per level:

$$\Theta(n) \cdot \Theta(\log(n/k)) = \Theta(n \log(n/k))$$

$\Theta(n)$ → work per level

$\Theta(\log(n/k))$ → number of levels

(c) (10 points) By Parts (a) and (b), the hybrid sorting algorithm runs in worst-case time $\Theta(nk + n\log(n/k))$. We would like to choose k as a function of n so that the worst-case order of growth for this hybrid algorithm is no worse than the order of growth for merge sort. What is the fastest rate of growth of k for which this holds?

$\Theta(nk + \log(n/k) = \Theta(nk + n(\log n - \log k)) = \Theta(nk + n\log n - n\log k)$

We know k cannot be larger than $\log n$ because then we will have a time complexity that is larger than $n\log n$ due to the term nk inside our current time complexity.

Assuming $k = \log n$ we get $\Theta(n\log n + n\log n - n\log\log n)$

↳ grows slower than $n\log n$ so we can remove it

$\Rightarrow \Theta(2n\log n)$

Dropping constants $\rightarrow \Theta(n\log n)$

k must also be greater than 1 for if k was 1 then we would have the original mergesort

$\therefore 1 < k \leq \log n$