1. (30 Points) Design a variation of MergeSort which, instead of recurring until the subarray has length 0 or 1, recurs at most a constant $k$ times, then calls InsertionSort on the resulting $2^k$ subarrays of length $n/2^k$. Call this algorithm Depth-$k$-MergeSort.

For Example let $q = \left\lfloor \frac{1+n}{2} \right\rfloor$, $u = \left\lfloor \frac{1+q}{2} \right\rfloor$, and $v = \left\lfloor \frac{(q+1)+n}{2} \right\rfloor$. We can picture the operation of Depth-2-MergeSort by the following recursion tree.

The nodes at depth $i$ represent subarrays of length $n/2^i$. Depth-2-MergeSort calls InsertionSort on the 4 subarrays at Depth 2. Depth-1-MergeSort recurs down to depth 1 and calls InsertionSort on the 2 subarrays at Depth 1. Depth-0-MergeSort simply calls InsertionSort on the full array.

a. (10 Points) Write pseudo-code for this algorithm. (Hint: Notice that each call to Depth-$k$-MergeSort must know its own level in the recursion tree in order to know whether it should recur again, or call InsertionSort.)

b. (10 Points) Determine the asymptotic run time of Depth-$k$-MergeSort as a function of both $k$ and $n$. In order to simplify the analysis, you may assume that $n$ is always an exact power of 2. (Hint: First review the discussion of InsertionSort in the text, and recall that InsertionSort runs in time $\Theta(n^2)$.)

c. (10 Points) Determine $k$ as a function of $n$ such that nothing is gained in terms of asymptotic run time by recurring more than $k$ times. (Hint: You will see from your answer to part (b) that if $k$ is a fixed constant, then Depth-$k$-MergeSort runs in time $\Theta(n^2)$). Observe that $k = \lg(n)$ gives ordinary MergeSort, which recurs down the level at which all arrays are of length 1, and runs in time $\Theta(n\lg(n))$. You are to determine a function $k = g(n)$ (which is less than $\lg(n)$ in absolute terms) such that Depth-$k$-MergeSort also runs in time $\Theta(n\lg(n))$.)