In general, merge insertion can be summarized as follows: Let the items to be sorted be \( \text{SORTED}[1 : n] \). Make pairwise comparisons of \( \text{SORTED}[i] \) and \( \text{SORTED}[i + 1] \); place the larger items into an array \( \text{HIGH} \) and the smaller items into array \( \text{LOW} \). If \( n \) is odd, then the last item of \( \text{SORTED} \) is appended to \( \text{LOW} \). Now apply merge insertion to the elements of \( \text{HIGH} \). Permute \( \text{LOW} \) using the same permutation. Then we know that \( \text{HIGH}[1] < \text{HIGH}[2] < \ldots < \text{HIGH}[\lfloor n/2 \rfloor] \) and \( \text{LOW}[i] < \text{HIGH}[i] \) for \( 1 \leq i \leq \lfloor n/2 \rfloor \). Now we insert the items of \( \text{LOW} \) into the \( \text{HIGH} \) array using binary insertion. However, the order in which we insert the \( \text{LOW} \)'s is important. We want to select the maximum number of items in \( \text{LOW} \) so that the number of comparisons required to insert each one into the already sorted list is a constant \( j \). As we have seen from our example, the insertion proceeds in the order \( \text{LOW}(t_j), \text{LOW}(t_j - 1), \ldots, \text{LOW}(t_j - 1) \), where the \( t_j \) are a set of increasing integers. In fact \( t_j \) has the form \( t_j = 2^j - t_j - 1 \), and in the exercises it is shown that this recurrence relation can be solved to give the formula \( t_j = \frac{(2^{j+1} + (-1)^j)}{3} \). Thus items are inserted in the order \( \text{LOW}[3], \text{LOW}[2]; \text{LOW}[5], \text{LOW}[4]; \text{LOW}[11], \text{LOW}[10], \text{LOW}[9], \text{LOW}[8], \text{LOW}[7], \text{LOW}[6] \); and so on.

It can be shown that the time for this algorithm is

\[
\sum_{1 \leq k \leq n} \left( \log_2 \frac{3k}{4} \right)
\]

(10.3)

For \( n = 1 \) to 21, the values of this sum are

0, 1, 3, 5, 7, 10, 13, 16, 19, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66

Comparing these values with \( \lceil \log n! \rceil \), we see that merge insertion is truly optimal for \( 1 \leq n \leq 11 \) and \( n = 20, 21 \).

### 10.1.3 Selection

From our previous discussion it should be clear that any comparison tree that models comparison-based algorithms for finding the maximum of \( n \) elements has at least \( 2^{n-1} \) external nodes since each path from the root to an external node must contain at least \( n - 1 \) internal nodes. This implies at least \( n - 1 \) comparisons for otherwise at least two of the input items never lose a comparison and the largest is not yet found.

Now suppose we let \( L_k(n) \) denote a lower bound for the number of comparisons necessary for a comparison-based algorithm to determine the largest, second largest, \( \ldots \), \( k \)th largest out of \( n \) elements, in the worst case. \( L_1(n) = n - 1 \) as previously. Since the comparison tree must contain enough external nodes to allow for any possible permutation of the input, it follows immediately that \( L_k(n) \geq \lceil \log n(n-1) \cdots (n-k+1) \rceil \). → a trivial bound

\( n(n-1) \cdots (n-k+1) \) = \( k \) largest elements 的可能位置数目
Theorem 10.2 \( L_k(n) \geq n - k + \lfloor \log n(n-1) \cdots (n-k+2) \rfloor \) for all integers \( k \) and \( n \), where \( \frac{n}{2} \leq k \leq n \).

Proof: As before internal nodes of the comparison tree contain integers of the form \( i : j \) that imply a comparison between the input items \( A[i] \) and \( A[j] \). If \( A[i] < A[j] \), then the algorithm proceeds down the left branch; otherwise it proceeds down the right branch. Now consider the set of all possible inputs and place inputs into the same equivalence class if their \( k - 1 \) largest values appear in the same positions. There will be \( n(n-1) \cdots (n-k+2) \) equivalence classes which we denote by \( E_i \), \( i = 1, 2, \ldots \). Now consider the external nodes for the set of inputs in the equivalence class \( E_i \) (for some \( i \)). The external nodes of the entire tree are also partitioned into classes called \( X_i \). For all external nodes in \( X_i \) the positions of the largest, \ldots, \( k-1 \)st-largest elements are identical. If we examine the subtree of the original comparison tree that defines the class \( X_i \), then we observe that all comparisons are made on the position of the \( n - k + 1 \) smallest elements; in essence we are trying to determine the \( k \)th-largest element. Therefore this subtree can be viewed as a comparison tree for finding the largest of \( n - k + 1 \) elements and it has at least \( 2^{n-k} \) external nodes. Hence the original tree contains at least \( n(n-1) \cdots (n-k+2)2^{n-k} \) external nodes and the theorem follows. \( \square \)

EXERCISES

1. Draw the comparison tree for sorting four elements.

2. Draw the comparison tree for sorting four elements that is produced by the binary insertion method.

3. When equality between keys is permitted, there are 13 possible permutations when sorting three elements. What are they?


5. Let \( TE(n) \) be the minimum number of comparisons needed to sort \( n \) items and to determine all equalities between them. It is clear that \( TE(n) \geq T(n) \) since the \( n \) items could be distinct. Show that \( TE(n) = T(n) \).

6. Find a comparison tree for sorting six elements that has all external nodes on levels 10 and 11.