The running time of the algorithm is the sum of running times for elements executed; a statement that takes \( c_i \) steps to execute and executes \( n \) elements contributes \( c_i n \) to the total running time. To compute \( T(n) \), the running time of \textsc{insertion-sort} on an input of \( n \) values, we sum the products of the \textit{cost} and \textit{times} columns, obtaining

\[
T(n) = c_1 n + c_2 (n - 1) + c_4 (n - 1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n - 1).
\]

Even for inputs of a given size, an algorithm’s running time may depend on which input of that size is given. For example, in \textsc{insertion-sort}, the best-case occurs if the array is already sorted. For each \( j = 2, 3, \ldots, n \), we have that \( A[i] \leq \text{key} \) in line 5 when \( i \) has its initial value of \( j - 1 \). Thus \( t_j = j \) for \( j = 2, 3, \ldots, n \), and the best-case running time is

\[
T(n) = c_1 n + c_2 (n - 1) + c_4 (n - 1) + c_5 (n - 1) + c_8(n - 1) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).
\]

We can express this running time as \( an + b \) for constants \( a \) and \( b \) that depend on the statement costs \( c_i \); it is thus a linear function of \( n \).

If the array is in reverse sorted order—that is, in decreasing order—then worst-case results. We must compare each element \( A[j] \) with each element in the already sorted subarray \( A[1 \ldots j - 1] \), and so \( t_j = j \) for \( j = 2, 3, \ldots, n \). Noting

\footnote{This characteristic does not necessarily hold for a resource such as memory. A statement that references \( m \) words of memory and is executed \( n \) times does not necessarily reference \( m \) words of memory.}
Suppose there are $N$ leaves, how large can height be?

- $h = 0$: 1 leaf
- $h = 1$: 2 leaves
- $h = 2$: 4 leaves
- $h = 3$: at most $2^h$ leaves

\[
\log n! = \sum_{i=1}^{n} \log i = \int_{1}^{n} \log x \, dx
\]

\[
= n \log n - n + O(1)
\]
External Path Length of a Tree $T = \text{EPL}(T)$

$$\text{EPL}(T) = \sum_{\text{leaves}} \text{depth}(l)$$

Any sorting time $\frac{n}{n!} \cdot \text{EPL(} \text{any tree for } n \text{ entries)}$

$\geq \ ?$

EPL (tree with $n!$ leaves) $\geq \ ?$

$N$ levels

$\frac{1}{n!} \cdot \text{EPL(} \text{full tree) } \geq N \log N$

$\text{EPL(} \text{full tree) } \geq n! \log n!$

$\text{Any sorting time } \geq \log n! = n \log n + O(\cdots)$
What does the tree $T$ of $N$ leaves w/ least possible EPL look like?

**Lemma:** The binary tree w/ min EPL has all of its leaves at level $k$ or $k+1$ for some value of $k$.

$\begin{array}{c}
\text{E}_{\text{L}+1} \\
\downarrow \\
\text{E}_{\text{L}}
\end{array}$

$\text{E}_{\text{L}} = \frac{1}{2} \text{E}_{\text{L}+1} + (L-1)$

$\Delta \text{E}_{\text{L}} = -L + L + 1 \leq 0 \quad \text{NO}$

**Proof:**

By contradiction — suppose there are leaves at $L$ and $L > L+1 \Rightarrow -L + L + 1 < 0$

$\begin{array}{c}
\text{L} \\
\downarrow \\
\text{E}_{\text{L}+1}
\end{array}$

Kraft's inequality

**Lemma:** Let $L_1, L_2, \ldots, L_N$ be the depths of the $N$ leaves in a binary tree. Then

$\sum_{i=1}^{N} 2^{-L_i} \leq 1$

**Proof:** Induction — you do it!
The min average starting from one of comparison $N = n!$

\[ N \geq 2^l \]
\[ 2^{e+1} > N \]

\[ e = \left\lfloor \log_2 N \right\rfloor \]

\[ h \leq e + 1 \]

\[ EPC = h \cdot k + (N-k)(e+1) \]

\[ = N \log N + O(1) \]

Satisfying $EPC \geq N \log N = n! \log n!$

\[ \frac{EPC}{n!} \geq \log n! = n \log n + O(1) \]

Then Satisfy takes time $\Omega(n \log n)$ in the worst case and on the average.
2 clubs
3 clubs

O(n) runtime

ace of spades