Lower Bounds for Comparison-Based Sorting Algorithms

Ekkapot Charoenwanit
Efficient Algorithms

September 6, 2020

1 Comparison-Based Sorting

In a comparison-based sorting algorithm, we gain information on relative order between the elements of a sequence $< a_1, a_2, ..., a_n >$ only through pair-wise comparisons. In other words, for any two elements a_i and a_j, we perform one of the following comparison tests

- $a_i < a_j$
- $a_i \leq a_j$
- $a_i > a_j$
- $a_i \geq a_j$
- $a_i = a_j$

to gain order information for a_i and a_j.

We will assume that all the elements are distinct without loss of generality. With this assumption, we can discard comparisons of the form $a_i = a_j$. Moreover, it follows that comparisons of the forms $a_i < a_j$, $a_i \leq a_j$, $a_i > a_j$ and $a_i \geq a_j$ are all equivalent in the sense that they provide us with the same information on their relative order. Therefore, we can assume that all comparisons made in comparison-based sorting algorithms are of the form $a_i \leq a_j$.

2 Decision Tree

All comparison-based sorting algorithms such as heapsort, mergesort, insertion sort etc. can be viewed in terms of decision trees. A decision tree is a full binary tree [1] where each internal node corresponds to a pair-wise comparison between two elements of the form $a_i \leq a_j$. Each of the $n!$ permutations on the original input sequence $< a_1, a_2, ..., a_n >$ must appear as one of the leaves of the decision tree. Depending on the input sequence $< a_1, a_2, ..., a_n >$, the execution of a sorting algorithm corresponds to following a simple path from the root down to a leaf.

Each internal node is denoted as $i : j$ to indicate a comparison between a_i and a_j of the form $a_j \leq a_j$, where a_i and a_j are the elements in positions i and j of the original input sequence $< a_1, a_2, ..., a_n >$. A leaf node is denoted as some permutation $< a_{\pi(1)}, a_{\pi(2)}, ..., a_{\pi(n)} >$ on the original input sequence.

Therefore, to accommodate all the possible $n!$ outcomes of any comparison-based sorting algorithm, the corresponding decision tree must have a sufficient number of leaves. That is, $l \geq n!$, where l denotes the number of leaves of the corresponding decision tree. Figure 1 shows the decision tree corresponding to running Insertion Sort on a sequence of 3 elements.

[1] A full binary tree is a binary tree in which every internal node has two children.

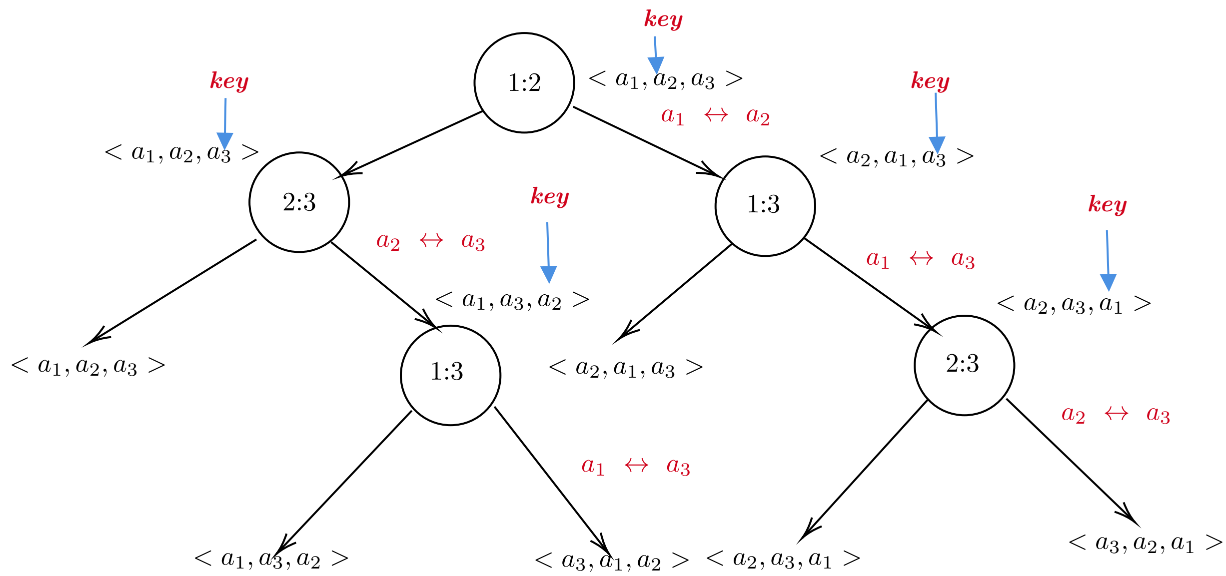

Figure 1: The decision tree corresponding to executing Insertion Sort on $A[1...3]$

Theorem 1. *Any binary tree of height h has at most 2^h leaves.*

Proof: We will prove by induction on the height h.

Base Case: $h = 0$

A binary tree of height has either 0 or 1 node $\leq 2^0 = 1$. ✓

Induction Hypothesis:

Assume true for $h = 0, 1, 2, ..., k - 1$.

Inductive Step:

Construct a binary tree T with height k. Let us denote the left subtree and the right subtree as T_L and T_R, respectively. Suppose T_L and T_R are of height h_L and h_R, respectively. Since T is of height k, $h_L \leq k-1$ and $h_R \leq k - 1$.

$$l_L \leq 2^{h_L} \quad \texttt{[I.H.]}$$

$$2^{h_L} \leq 2^{k-1} \quad \texttt{[}h_L \leq k-1\texttt{]}$$

$$l_L \leq 2^{k-1} \quad \texttt{[Transitivity of } \leq \texttt{]} \tag{1}$$

$$l_L \leq 2^{h_R} \quad \texttt{[I.H.]}$$

$$2^{h_R} \leq 2^{k-1} \quad \texttt{[}h_R \leq k-1\texttt{]}$$

$$l_R \leq 2^{k-1} \quad \texttt{[Transitivity of } \leq \texttt{]} \tag{2}$$

$$l_L + l_R \leq 2^{k-1} + 2^{k-1} \texttt{[Adding Eq.(1) and Eq.(2)]}$$

$$l_L + l_R \leq 2^k \quad \texttt{[}2^k = 2^{k-1} + 2^{k-1}\texttt{]}$$

$$l \leq 2^k \quad \texttt{[}l = l_L + l_R\texttt{]}$$

Conclusion: We have just shown that any binary tree of height h has at most 2^h leaves. $\square$

Theorem 2. *The running time of any comparison-based sorting algorithm is $\Omega(n \log n)$, where n is the length of the input sequence.*

Proof: By Theorem 1, it follows that $2^h \geq l \geq n!$. That is, $2^h \geq n!$.

$$h \geq \log n! \quad \texttt{[Taking } \log \texttt{ on both sides]}$$

$$h \geq \log n + \log(n-1) + ... + \log 1 \quad \texttt{[Expanding } \log n!\texttt{]}$$

$$\log n + \log(n-1) + ... + \log 1 \geq \log \frac{n}{2} + \log(\frac{n}{2} + 1) + ... + \log n \quad \texttt{[Observation \& Verification]}$$

$$\log \frac{n}{2} + \log(\frac{n}{2} + 1) + ... + \log n \geq \frac{n}{2} \cdot \log \frac{n}{2} \quad \texttt{[Observation \& Verification]}$$

$$h \geq \frac{n}{2} \cdot \log \frac{n}{2} \quad \texttt{[Transitivity of } \geq\texttt{]}$$

Since the height h is determined by a longest simple path from the root down to a leaf, the number of comparisons/swaps in the worst case is h, which is at least $\frac{n}{2} \log \frac{n}{2}$. Since the running time $T(n)$ of a sorting algorithm is determined by the number of comparisons/swaps, $T(n) \in \Omega(\frac{n}{2} \log \frac{n}{2}) = \Omega(n \log n)$. $\square$