Sorting

S is a finite linearly ordered set
L is an initial list of all elements of S
in an arbitrary order. A sorting is a reordering L into L' where all the elements are in increasing order.

2, 4, 1, 5, 3 \rightarrow 1, 2, 3, 4, 5

The key question is the complexity of sorting.

**Binary Comparison** = the comparison of two elements at a time.

Two possible outcomes of each comparison: "a < b" or "a > b".

2-Branching at each non-terminal vertex of a decision tree. **Decision Tree is Binary**.

# of leaves = # of all possible inputs = # of permutations of n elements = n!

**In fact,** n! leaves

Instead, # of leaves = $2^n$. **Worst-case complexity = n height of decision tree**.

The Bubble Sort

Consistently move the whole list.
Interchange a larger element with a smaller one following it.

\[
\begin{array}{cccc}
3 & 3 & 3 & 3 \\
1 & 2 & 2 & 2 \\
4 & 2 & 2 & 4 \\
2 & 4 & 2 & 2 \\
1 & 2 & 4 & 4 \\
\end{array}
\]

Pass #1, guarantees the last position is correct.

Pass #2, guarantees the last two positions (at the least).

Complexity = # of comparisons = \((n-1) + (n-2) + \ldots + 2 + 1 = \frac{6}{2} = O(n^2)\)

Does not reach \(O(n \log n)\) complexity.
Simple but not fast!

The Merge Sort

Sorted lists \(L_1\) and \(L_2\).

Faster than to sort \(n\) list.

Method: compare the smallest elements of two remaining lists \(L_1, L_2\), add the smallest of two to the end of the list \(L\).

\[
\begin{array}{c}
2 \downarrow \\
43 \downarrow \\
6 \downarrow \\
13 \downarrow \\
2v1 \\
2v3 \\
4v3 \\
4v7 \\
5v7 \\
6v7 \\
\end{array}
\]

Lemma: the above algorithm merges two sorted lists of \(m\) and \(n\) elements using not more than \(m+n-1\) comparisons.

Proof: each comparison reduces the combined length of \(L_1, L_2\) by 1, no comparisons needed when the combined length = 1.
The Merge Sort Algorithm: Recursive Definition

1. Split a list into two sublists of (approximately) equal size.
2. Sort each part by the Merge Sort algorithm.
3. Merge two sorted parts.

The number of comparisons needed to sort a list by the Merge Sort algorithm is \(O(n \log n)\).

Proof: Assume \(n = 2^m\) to simplify the argument (without loss of generality).

Without loss of generality:

1. List of \(2^m\) elements
2. Lists of \(2^{m-1}\) elements
3. Lists of \(2^{m-2}\) elements
   - \(2^{m-k}\) lists of \(2^{m-k}\) elements
   - \(2^m\) lists of 1 element

Comparisons on the bottom level: On level \(k\):
\[2^k = 2^{m-k}\] merges of pairs of \(2^{m-k}\) lists = \(2^{m-k}(2^{m-k}-1)\) comparisons.

\[
N = \sum_{k=1}^{m} 2^k(2^{m-k}-1) = \sum_{k=1}^{m} 2^k(2^{m-k}) = m2^m - (2^m - 1) = n \log n - n + 1
\]

\(O(n \log n)\)