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 → 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 "b < a"

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!

\[ \log n! \leq n \leq 2^n \]

Indeed,

\[ \log n! \leq n \]
**Th.1.** Any sorting algorithm based on binary comparisons requires at least \( \Omega \log n! \) comparisons.

**Cor.** No sorting algorithm that uses comparisons as the method of sorting can have worst-case complexity that is better than \( \Theta(n \log n) \).

**Proof.**

\[ n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \]

\[ \frac{n!}{n^2} \geq \frac{(n-1)!}{(n-2)^2} \cdot \frac{1}{n} \geq \ldots \geq \frac{2!}{2^2} \cdot \frac{1}{n} = \frac{n!}{n^2} \]

\[ \log \left( \frac{n!}{n^2} \right) \geq \log \left( \frac{(n-1)!}{(n-2)^2} \cdot \frac{1}{n} \right) \geq \ldots \geq \log \left( \frac{2!}{2^2} \cdot \frac{1}{n} \right) = \log \frac{n!}{n^2} \]

\[ \frac{n}{2} \log \frac{n}{2} \leq \log \left( \frac{n}{2} \right) \cdot \log \frac{n}{2} \leq \log \frac{n}{2} \leq \log n ! \leq n \cdot \log n \]

\[ \log \frac{n}{2} > \log \sqrt{n}, \text{ since } \frac{n}{2} > \sqrt{n} \text{ for } n > 4 \]

\[ \frac{n}{2} \log \frac{n}{2} > \frac{1}{2} \log n = \frac{1}{2} \cdot \log n \frac{1}{2} = \frac{n}{2} \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} \]

\[ \frac{n}{2} \log \frac{n}{2} < \log n ! < n \cdot \log n \quad (\text{for } n > 4) \]
THE BUBBLE SORT

Consequent passes of the whole list interchanging a larger element with a smaller one following it.

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

Pass # One,
Guarantees the last position is correct!

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

Pass # Two,
Guarantees the last two positions (at the least)

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

Pass # Tree

Complexity = \# of comparisons =
\[
= (n-1) + (n-2) + \cdots + 2 + 1 = \frac{(n-1) \cdot n}{2} = \mathcal{O}(n^2)
\]

Does not reach \(\mathcal{O}(n \log n)\) complexity.
Simple but not fast!
THE MERGE SORT

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

Their sorted merge \( L \)

\[ L = \begin{aligned} L_1 \quad \text{\( n \)} \quad \text{L}_2 \quad \text{\( m \)} \quad \text{\( n + m \)} \end{aligned} \]

Faster than to sort \( n + m \) list

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

\( L_1: 2 \; 4 \; 5 \; 6 \) \( L_2: 1 \; 3 \; 7 \) \( L: 1 \; 2 \; 3 \; 4 \; 5 \; 6 \; 7 \)

Comparisons:
- \( 2 \; v \; 1 \): 1
- \( 2 \; v \; 1 \): 2
- \( 4 \; v \; 3 \): 2
- \( 4 \; v \; 3 \): 3
- \( 4 \; v \; 7 \): 4
- \( 5 \; v \; 7 \): 5
- \( 6 \; v \; 7 \): 6

If one of the lists is empty, add the remaining one to the end of the merge.

Lemma: The above algorithm merges two sorted lists of \( n \) and \( m \) elements using not more than \( n + m - 1 \) 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 $O(n \cdot \log n)$

PROOF. ASSUME $n = 2^m$ TO SIMPLIFY THE ARGUMENT (WITHOUT LOSS OF GENERALITY)

$$N = \sum_{k=1}^{m} 2^k \cdot \left( \frac{2^m}{2^k} - 1 \right) = \sum_{k=1}^{m} \left( 2^m - 2^k \right) = m \cdot 2^m - \left( 2^m - 1 \right) = n \cdot \log n - n + 1$$