What is a Lower Bound?

- Provides information about the best possible efficiency of ANY algorithm for a problem
- Tells us whether we can improve an algorithm or not
- When lower bound is (asymptotically) the same as the best upper bound (provided by an algorithm), the bound is TIGHT
- If there is a gap, there might be room for improvement
Techniques for Proving Lower Bounds

- Trivial (size of input or output)
  - Ex: Any algorithm to generate all permutations of $n$ elements requires time $\Omega(n!)$ because there are $n!$ outputs
  - Ex: Computing the product of two $n$-by-$n$ matrices requires time $\Omega(n^2)$ because the output has $n^2$ entries
  - Ex: Any TSP solution for $n$ cities requires $\Omega(n^2)$ time because there are $\Theta(n^2)$ inputs to be taken into account.
Techniques for Proving Lower Bounds

- Information-theoretic:
  - Consider the amount of information that the solution must provide
  - Show that each step of the algorithm can only produce so much information
  - Uses mechanism of “decision trees”
  - Example next...
Comparison-Based Sorting

- We’ve seen mergesort, insertion sort, quicksort, heapsort,...
- All these algorithms are comparison-based
  - the behavior depends on relative values of keys, not exact values
  - behavior on [1,3,2,4] is same as on [9,25,23,99]
- Fastest of these algorithms was $O(n \log n)$.
- We will show that's the best you can get with comparison-based sorting.
Decision Tree

- Consider *any* comparison based sorting algorithm
- Represent its behavior on all inputs of a fixed size with a *decision tree*
- Each tree node corresponds to the execution of a comparison
- Each tree node has two children, depending on whether the parent comparison was true or false
- Each leaf represents correct sorted order for that path
Decision Tree Diagram

first comparison: check if \( a_i \leq a_j \)

- YES
  - second comparison
    - if \( a_i \leq a_j \): check if \( a_k \leq a_l \)
      - YES
        - third comparison
          - if \( a_i \leq a_j \) and \( a_k \leq a_l \): check if \( a_x \leq a_y \)
        - NO
      - NO
    - NO
  - NO
- NO

- second comparison
  - if \( a_i > a_j \): check if \( a_m \leq a_p \)
    - YES
      - NO
    - NO

Insertion Sort

for j := 2 to n to
    key := a[j]
    i := j-1
    while i > 0 and a[i] > key do
        a[i+1] := a[i]
        i := i -1
    endwhile
    a[i+1] := key
endfor
Insertion Sort for $n = 3$

Diagram showing the decision process for insertion sort with $n = 3$. The process starts with comparing $a_1$ and $a_2$, then $a_2$ and $a_3$, and finally $a_1$ and $a_3$. Each comparison leads to a decision path that results in the sorted order $a_1$, $a_2$, $a_3$. The possible outcomes are $a_1 \leq a_2$, $a_2 \leq a_3$, and $a_1 \leq a_3$. The sorted sequences are $a_1 a_2 a_3$, $a_1 a_3 a_2$, $a_3 a_1 a_2$, $a_2 a_3 a_1$, and $a_3 a_2 a_1$. The decision tree visually represents these comparisons and outcomes.
Insertion Sort for $n = 3$
How Many Leaves?

- Must be at least one leaf for each permutation of the input
  - otherwise there would be a situation that was not correctly sorted
- Number of permutations of $n$ keys is $n!$.
- Idea: since there must be a lot of leaves, but each decision tree node only has two children, tree cannot be too shallow
  - depth of tree is a **lower bound** on running time
Key Lemma

Height of a binary tree with $n!$ leaves is $\Omega(n \log n)$.

Proof: Maximum number of leaves in a binary tree with height $h$ is $2^h$.

$h = 1$, $2^1$ leaves

$h = 2$, $2^2$ leaves

$h = 3$, $2^3$ leaves
Proof of Lemma

- Let $h$ be height of decision tree.
- Number of leaves in decision tree, which is $n!$, is at most $2^h$.

\[ 2^h \geq n! \]
\[ h \geq \log(n!) \]
\[ = \log(n(n-1)(n-2)...(2)(1)) \]
\[ \geq (n/2)\log(n/2) \quad \text{by algebra} \]
\[ = \Omega(n \log n) \]
Finishing Up

- Any binary tree with $n!$ leaves has height $\Omega(n \log n)$.
- Decision tree for any comparison-based sorting alg on $n$ keys has height $\Omega(n \log n)$.
- Any comp.-based sorting alg has at least one execution with $\Omega(n \log n)$ comparisons.
- Any comp.-based sorting alg has $\Omega(n \log n)$ worst-case running time.
Techniques for Proving Lower Bounds

**Problem reduction:**
- Assume we already know that problem P is hard (i.e., there is some lower bound of X on its running time)
- Show that problem Q is at least as hard as problem P by reducing problem P to Q
- I.e., if we can solve Q, then we can solve P
- Implies that X is also a lower bound for Q