Class 32

LOWER BOUNDS

DECISION TREES

Lower Bounds on Algorithm Efficiency

Problem complexity: worst-case running time necessary and sufficient to solve the problem

To show sufficient: give algorithm with that worst-case runtime

To show necessary: need a lower bound—a guarantee on a minimum worst-case runtime needed to solve the problem (via any algorithm)

- Harder to prove than algorithm efficiency. Universal statement

Can be exact count (say, #comps) or efficiency class (Ω)

Bound is tight if there exists an algorithm with the same efficiency as the bound (in which case, we've identified the problem complexity)

Methods for Establishing Lower Bounds

Trivial lower bounds

Information-theoretic arguments

Adversary arguments

Problem reduction

Trivial Lower Bounds

- number of inputs that must be processed (perhaps not all!)
- number of outputs that must be produced

Examples:

- An algorithm for finding the smallest (or largest) element in an unsorted array has to process all array elements.
- An algorithm producing all permutation of n distinct elements has to produce n! outputs.

Information-theoretic arguments

- Each step gains a limited amount of information to build the solution; lower bound is number of steps to accumulate enough information
- We assume the *decision-tree model of computation*, where branches (decisions) are made based only on the result of comparison operations.

Decision Tree for Sorting

Comparison based sorting can be formalized as a problem of classifying which permutation of *n* items is given to the sorting algorithm.

Number of permutations of *n* items: n!

To classify which permutation is given we need to ask questions, in particular yes/no questions.

We can use a binary decision tree to model this behavior.

Decision Tree: Selection Sort

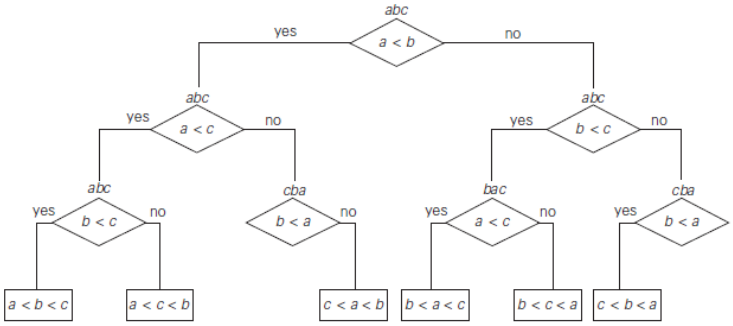


FIGURE 11.2 Decision tree for the tree-element selection sort. A triple above a node indicates the state of the array being sorted. Note two redundant comparisons $b < a$ with a single possible outcome because of the results of some previously made comparisons.

Decision Tree: Insertion Sort

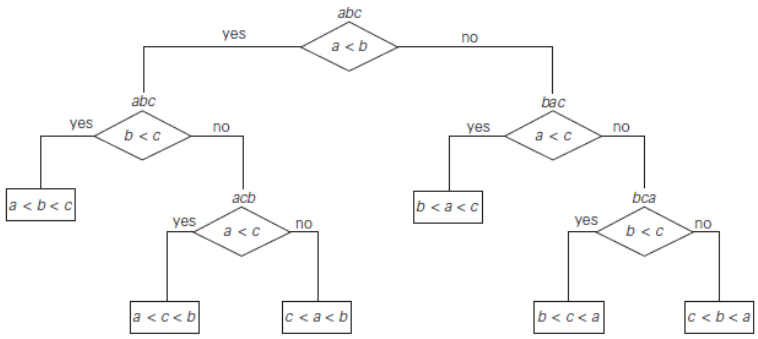


FIGURE 11.3 Decision tree for the three-element insertion sort.

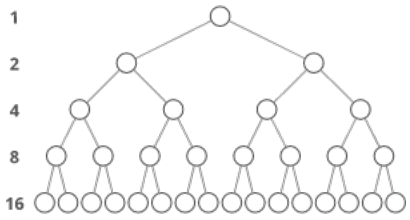
Decision Tree for Sorting

How large does the tree have to be?

For starters, it need to accommodate n! leaves.

If *h* is the height of the tree, then 2^h >= n! otherwise there is not enough room in the inn.

The tree on the left has:
16 leaves
a height of 4.



Decision Tree for Sorting

And now, some fun with Math:

$2^h >= n!$

$h >= \log_2(n!)$

$= \log_2(n(n-1)(n-2) \ldots (2))$

$= \log_2(n) + \log_2(n-1) + \log_2(n-2) + \ldots + \log_2 2))$

$= \sum_{i=2}^{n} \log_2(i)$

$= \sum_{i=2}^{n/2-1} \log_2(i) + \sum_{i=n/2}^{n} \log_2(i)$

$>= 0 + \sum_{i=n/2}^{n} \log_2(n/2)$

$= n/2 * \log_2(n/2)$

$= \Omega(n \log_2 n)$

Adversary arguments

- Imagine a malevolent-but-honest adversary pushing the algorithm down the most time-consuming path
- Lower bound: amount of work to reduce problem size to constant along path

Example:

- "Guessing" a number between 1 and n with yes/no questions: Adversary puts the number in a larger of the two subsets generated by previous question

Adversary Arguments

Example: Merging two sorted lists of size n

$$a_1 < a_2 < \ldots < a_n \text{ and } b_1 < b_2 < \ldots < b_n$$

(1, 3, 5, 7) and (2, 4, 6, 8)

- Output $b_1 < a_1 < b_2 < a_2 < \ldots < b_n < a_n$ requires $2n - 1$ comparisons of adjacent elements

Problem reduction

- Some problem Q has a known lower bound
- If we can efficiently reduce/transform Q to P then P shares the same lower bound

Example:

1. *P* is finding Minimum Spanning Tree for *n* points in Cartesian plane
2. *Q* is element uniqueness problem, i.e. are the elements in a data-structure unique.
3. The element uniqueness problem is known to be in $\Omega(n \log n)$
4. Transform a set $x_1, x_2, \ldots, x_n$ of *n* real numbers into a set of *n* points in a Cartesian space. Do this by adding 0 as the point's *y* coordinate: $(x_1, 0)$, $(x_2, 0)$, …, $(x_n, 0)$. Cost: $\Omega(n)$
5. Let T be a minimum spanning tree found for this set of points. T must contain a shortest edge. Cost of locating it: $\Omega(n)$
6. If that edge has a length of 0, then the given elements are not unique.

Reduction Exercises

Show multiplication of two *n*-digit integers and squaring an *n*-digit integer are in the same complexity class.

What about the same question for multiplying vs. squaring square matrices?