

Algorithms and Data Structures: Lower Bounds for Sorting

14th October, 2011

Comparison Based Sorting Algorithms

Definition 1

A sorting algorithm is *comparison based* if comparisons $A[i] < A[j]$, $A[i] \leq A[j]$, $A[i] = A[j]$, $A[i] \geq A[j]$, $A[i] > A[j]$ are the only ways in which it accesses the input elements.

Comparison Based Sorting Algorithms

Definition 1

A sorting algorithm is *comparison based* if comparisons $A[i] < A[j]$, $A[i] \leq A[j]$, $A[i] = A[j]$, $A[i] \geq A[j]$, $A[i] > A[j]$ are the only ways in which it accesses the input elements.

Comparison based sorting algorithms are *generic* in the sense that they can be used for all types of elements that are *comparable* (such as objects of type Comparable in Java).

Comparison Based Sorting Algorithms

Definition 1

A sorting algorithm is *comparison based* if comparisons $A[i] < A[j]$, $A[i] \leq A[j]$, $A[i] = A[j]$, $A[i] \geq A[j]$, $A[i] > A[j]$ are the only ways in which it accesses the input elements.

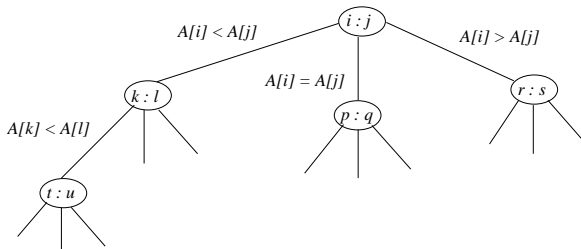
Comparison based sorting algorithms are *generic* in the sense that they can be used for all types of elements that are *comparable* (such as objects of type Comparable in Java).

Example 2

INSERTION-SORT, QUICKSORT, MERGE-SORT, HEAPSORT are all comparison based.

The Decision Tree Model

Abstractly, we may describe the behaviour of a comparison-based sorting algorithm S on an input array $A = \langle A[1], \dots, A[n] \rangle$ by a decision tree:

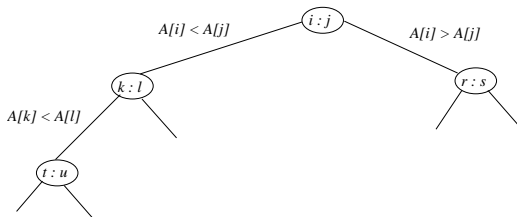


At each leaf of the tree the output of the algorithm on the corresponding execution branch will be displayed. Outputs of sorting algorithms correspond to permutations of the input array.

A Simplifying Assumption

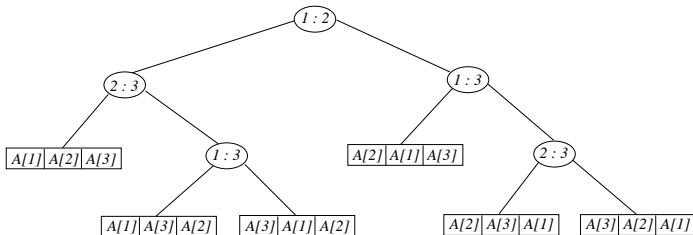
*In the following, we assume that all keys of elements of the input array of a sorting algorithm are distinct. (It is ok to restrict to a special case, because we want to prove a **lower bound**.)*

Thus the outcome $A[i] = A[j]$ in a comparison will never occur, and the decision tree is in fact a binary tree:



Example

Insertion sort for $n = 3$:



In insertion sort, when we get the result of a comparison, we often swap some elements of the array. In showing decision trees, we don't *implement* a swap. Our indices always refer to the *original* elements at that position in the array. To understand what I mean, draw the evolving array of INSERTIONSORT beside this decision tree.

A Lower Bound for Comparison Based Sorting

For a comparison based sorting algorithm S :

$C_S(n)$ = worst-case number of comparisons performed
by S on an input array of size n .

A Lower Bound for Comparison Based Sorting

For a comparison based sorting algorithm S :

$C_S(n)$ = worst-case number of comparisons performed
by S on an input array of size n .

Theorem 3

For all comparison based sorting algorithms S we have

$$C_S(n) = \Omega(n \lg n).$$

A Lower Bound for Comparison Based Sorting

For a comparison based sorting algorithm S :

$C_S(n)$ = worst-case number of comparisons performed
by S on an input array of size n .

Theorem 3

For all comparison based sorting algorithms S we have

$$C_S(n) = \Omega(n \lg n).$$

Corollary 4

The worst-case running time of any comparison based sorting algorithm is $\Omega(n \lg n)$.

A Lower Bound for Comparison Based Sorting

Proof of Theorem 3 uses Decision-Tree Model of sorting.

It is an **Information-Theoretic Lower Bound**:

A Lower Bound for Comparison Based Sorting

Proof of Theorem 3 uses Decision-Tree Model of sorting.

It is an **Information-Theoretic Lower Bound**:

- ▶ “Information-Theoretic” means that it is based on the amount of “information” that an instance of the problem can encode.

A Lower Bound for Comparison Based Sorting

Proof of Theorem 3 uses Decision-Tree Model of sorting.

It is an **Information-Theoretic Lower Bound**:

- ▶ “Information-Theoretic” means that it is based on the amount of “information” that an instance of the problem can encode.
- ▶ For sorting, the input can encode $n!$ outputs.

A Lower Bound for Comparison Based Sorting

Proof of Theorem 3 uses Decision-Tree Model of sorting.

It is an **Information-Theoretic Lower Bound**:

- ▶ “Information-Theoretic” means that it is based on the amount of “information” that an instance of the problem can encode.
- ▶ For sorting, the input can encode $n!$ outputs.
- ▶ Proof does not make *any* assumption about *how* the sorting might be done (except it is comparison-based).

Proof of Theorem 3

Observation 5

For every n , $C_S(n)$ is the height of the decision tree of S on inputs n (the longest path from the “root” to a leaf is the maximum number of comparisons that algorithm S will do on an input of length n).

We shall prove a lower bound for the height of the decision tree for *any* algorithm S .

Remark

Maybe you are wondering . . . was it **really** ok to assume all keys are distinct?

Proof of Theorem 3

Observation 5

For every n , $C_S(n)$ is the height of the decision tree of S on inputs n (the longest path from the “root” to a leaf is the maximum number of comparisons that algorithm S will do on an input of length n).

We shall prove a lower bound for the height of the decision tree for *any* algorithm S .

Remark

Maybe you are wondering . . . was it **really** ok to assume all keys are distinct?

It is ok - because the problem of sorting n keys (with no distinctness assumption) is *more general* than the problem of sorting n distinct keys.

The worst-case for sorting certainly is as bad as the worst-case for all-distinct keys sorting.

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.
- ▶ Putting everything together, we get

$$n! \leq \text{number of leaves of decision tree}$$

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.
- ▶ Putting everything together, we get

$$\begin{aligned} n! &\leq \text{number of leaves of decision tree} \\ &\leq 2^{\text{height of decision tree}} \end{aligned}$$

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.
- ▶ Putting everything together, we get

$$\begin{aligned} n! &\leq \text{number of leaves of decision tree} \\ &\leq 2^{\text{height of decision tree}} \\ &\leq 2^{C_S(n)}. \end{aligned}$$

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.
- ▶ Putting everything together, we get

$$\begin{aligned} n! &\leq \text{number of leaves of decision tree} \\ &\leq 2^{\text{height of decision tree}} \\ &\leq 2^{C_S(n)}. \end{aligned}$$

- ▶ Thus

$$C_S(n) \geq \lg(n!) = \Omega(n \lg(n)).$$

Observation 6

Each permutation of the inputs must occur at at least one leaf of the decision tree.

(Obs 6 *must* be true, if our algorithm is to sort properly for *all* inputs.).

- ▶ By Obs 6, the decision tree on inputs of size n has at least $n!$ leaves (for *any* algorithm S).
- ▶ The (simplified) decision tree is a binary tree. A binary tree of height h has at most 2^h leaves.
- ▶ Putting everything together, we get

$$\begin{aligned}n! &\leq \text{number of leaves of decision tree} \\ &\leq 2^{\text{height of decision tree}} \\ &\leq 2^{C_S(n)}.\end{aligned}$$

- ▶ Thus

$$C_S(n) \geq \lg(n!) = \Omega(n \lg(n)).$$

To obtain the last inequality, we can use the following inequality:

$$n^{n/2} \leq n! \leq n^n$$

This tells us that $\lg n! \geq \lg(n^{n/2}) = (n/2) \lg n = \Omega(n \lg(n))$.

Thm 3 QED

ADS: lect 7 – slide 24 – 14th October, 2011

An Average Case Lower Bound

For any comparison based sorting algorithm S :

$A_S(n)$ = average number of comparisons performed by S on an input array of size n .

Theorem 7

For all comparison based sorting algorithms S we have

$$A_S(n) = \Omega(n \lg n).$$

An Average Case Lower Bound

For any comparison based sorting algorithm S :

$A_S(n)$ = average number of comparisons performed by S on an input array of size n .

Theorem 7

For all comparison based sorting algorithms S we have

$$A_S(n) = \Omega(n \lg n).$$

Proof uses the fact that the average length of a path from the root to a leaf in a binary tree with ℓ leaves is $\Omega(\lg \ell)$ (Theorem 11 and 12).

Corollary 8

The average-case running time of any comparison based sorting algorithm is $\Omega(n \lg n)$.

Average root-leaf length in Binary tree

Definition 9

For any binary tree T , let $AvgRL(T)$ denote the *average root-to-leaf* length for T .

Definition 10

A *near-complete* binary tree T is a binary tree in which every internal node has *exactly* two child nodes, and all leaves are either at depth h or depth $h - 1$.

Theorem 11

Any “near-complete” binary tree T with leaf set $L(T)$, $|L(T)| \geq 4$, has Average *root-to-leaf* length $AvgRL(T)$ at least $\lg(|L(T)|)/2$.

Theorem 12

For any binary tree T , there is a near-complete binary tree T' such that $L(T) = L(T')$ (same leaf set) and such that $AvgRL(T') \leq AvgRL(T)$. Hence $AvgRL(T) \geq \lg(|L(T)|)/2$ holds for *all binary trees*.

Implications of These Lower Bounds

Theorem 3 and Theorem 7 are significant because they hold for *all* comparison-based algorithms S . They imply the following:

1. By Thm 3, any comparison-based algorithm for sorting which has a worst-case running-time of $O(n \lg n)$ is *asymptotically optimal* (ie, apart from the constant factor inside the “O” term, it is as good as possible in terms of worst-case analysis). This includes algorithms like MERGESORT, HEAPSORT.
2. By Thm 7, any comparison-based algorithm for sorting which has an average-case running-time of $O(n \lg n)$ is the best you can hope for in terms of average-case analysis (apart from the constant factor inside the “O” term). This is accomplished by MERGESORT, HEAPSORT and QUICKSORT (good in practice).

Lecture 8

We show how in a *special case* of sorting (when the inputs are numbers, coming from the range $\{1, 2, \dots, n^k\}$ for some constant k , we can sort **in linear time** (NOT a comparison-based algorithm).

Reading Assignment

[CLRS] Section 8.1 (2nd and 3rd edition) or

[CLR] Section 9.1

Well-worth reading - this is a nice chapter of CLRS (not too long).

Problems

1. Draw (simplified) decision trees for INSERTION SORT and QUICKSORT for $n = 4$.
2. Exercise 8.1-1 of [CLRS] (both 2nd and 3rd ed). *This is 9.1-1 of [CLR].*
3. Resolve the complexity (in terms of no-of-comparisons) of sorting 4 numbers.
 - 3.1 Give an algorithm which sorts any 4 numbers and which uses at most 5 comparisons in the worst-case.
 - 3.2 Prove (using the decision-tree model) that there is no algorithm to sort 4 numbers, which uses less than 5 comparisons in the worst-case.