Next: Lower bounds

Q: Can we beat the Ω(n log n) lower bound for sorting?

A: In general no, but in some special cases YES!

Ch 7: Sorting in linear time

Let's prove the Ω(n log n) lower bound.

Lower bounds

- What are we counting?

  Running time? Memory? Number of times a specific operation is used?
- What (if any) are the assumptions?
- Is the model general enough?

Here we are interested in lower bounds for the WORST CASE. So we will prove (directly or indirectly):

for any algorithm for a given problem, for each n>0, there exists an input that make the algorithm take $\Omega(f(n))$ time. Then f(n) is a lower bound on the worst case running time.

Comparison-based algorithms

Finished looking at comparison-based sorts.

Crucial observation: All the sorts work for any set of elements – numbers, records, objects,……

Only require a comparator for two elements.

```
#include <stdlib.h>
```

void qsort(void *base, size_t *nmemb*, size_t *size*, int(**compar*)(const void *, const void *));

DESCRIPTION: The qsort() function sorts an array with *nmemb* elements of *size* size. The base argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point to the objects being compared.

Comparison-based algorithms

- The algorithm only uses the results of comparisons, not values of elements (*).
- Very general – does not assume much about what type of data is being sorted.
- However, other kinds of algorithms are possible!
- In this model, it is reasonable to count #comparisons.
- Note that the #comparisons is a **lower bound** on the running time of an algorithm.

(*) If values are used, lower bounds proved in this model are not lower bounds on the running time.

Lower bound for a simpler problem

Let's start with a simple problem.

Minimum of n numbers

Minimum (A)

1. min = A[1]
2. for i = 2 to length[A]
3. do if min >= A[i]
4. then min = A[i]
5. return min

Can we do this with fewer comparisons?

We have seen very different algorithms for this problem. How can we show that we cannot do better by being smarter?

Lower bounds for the minimum

Claim: Any comparison-based algorithm for finding the minimum of n keys must use at least n-1 comparisons.

Proof: If x,y are compared and x > y, call x the winner.

Any key that is not the minimum must have won at least one comparison. WHY?

Each comparison produces exactly one winner and at most one NEW winner.

$\Rightarrow$ at least n-1 comparisons have to be made.

Points to note

Crucial observations: We proved a claim about ANY algorithm that only uses comparisons to find the minimum. Specifically, we made no assumptions about

1. Nature of algorithm.
2. Order or number of comparisons.
3. Optimality of algorithm
4. Whether the algorithm is reasonable – e.g. it could be a very wasteful algorithm, repeating the same comparisons.

On lower bound techniques

Unfortunate facts:

Lower bounds are usually hard to prove.

Virtually no known general techniques – must try ad hoc methods for each problem.

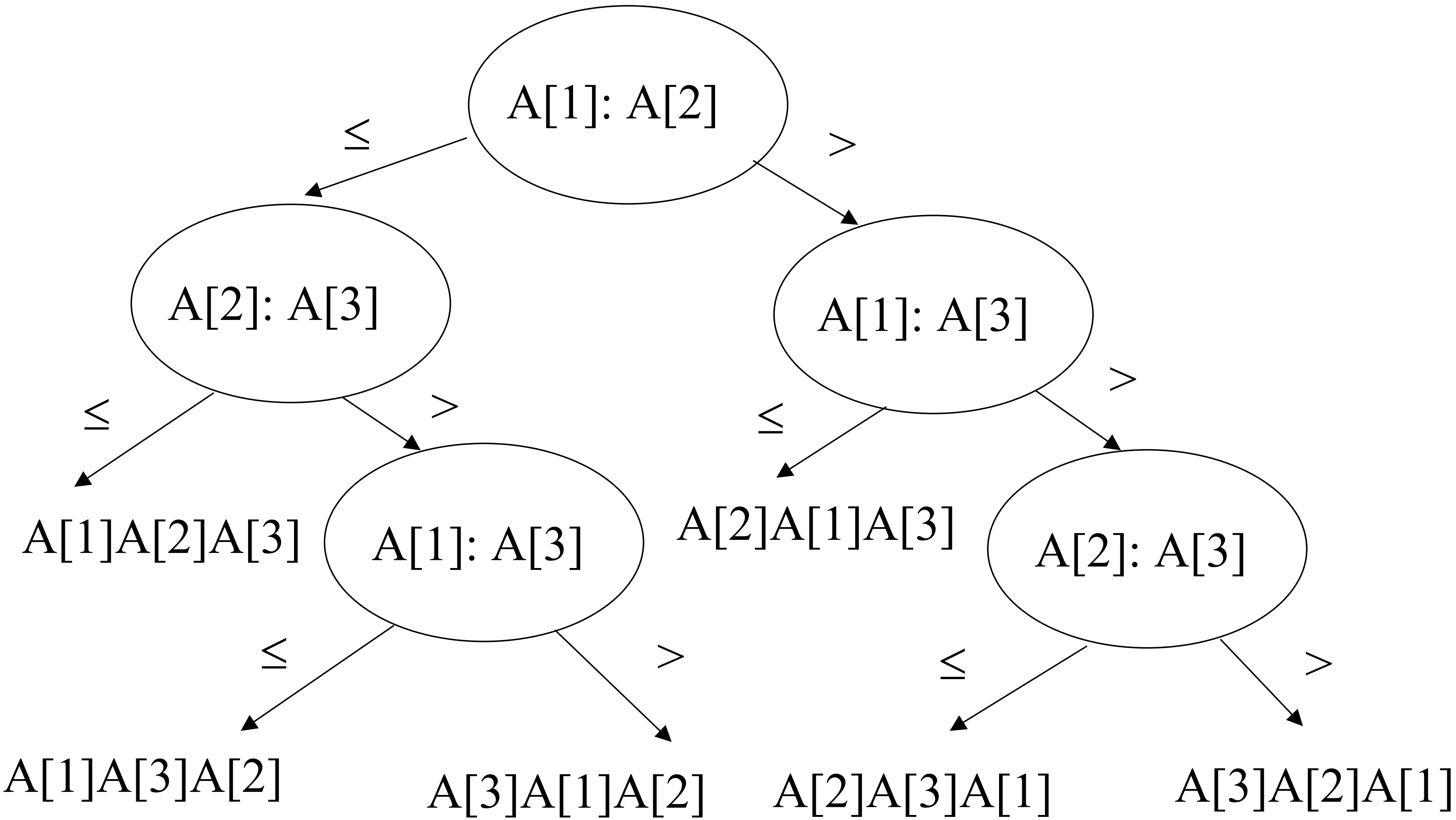
Lower bounds for comparison-based sorting

- Trivial: $\Omega(n)$ – every element must take part in a comparison.
- Best possible result – $\Omega(n \log n)$ comparisons, since we already know several O(n log n) sorting algorithms.
- Proof is non-trivial: how do we reason about all possible comparison-based sorting algorithms?

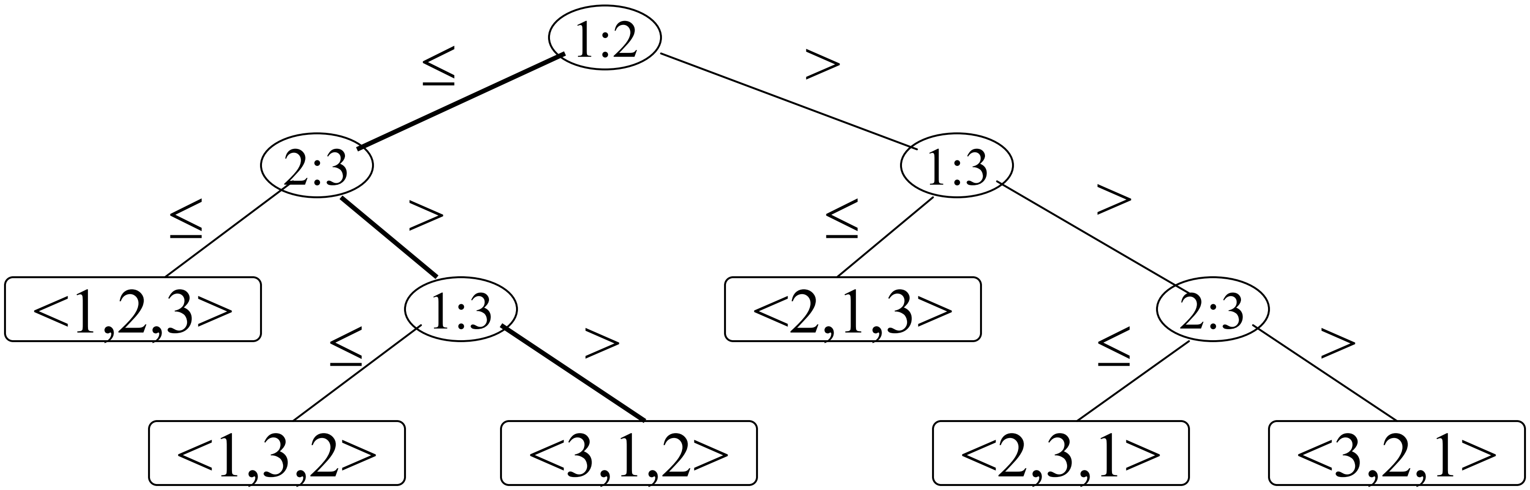
The Decision Tree Model

- Assumptions:
  - All numbers are distinct (so no use for $a_i = a_j$)
  - All comparisons have form $a_i \le a_j$ (since $a_i \le a_j$, $a_i \ge a_j$, $a_i < a_j$, $a_i > a_j$ are equivalent).
- Decision tree model
  - Full binary tree
  - Ignore control, movement, and all other operations, just use comparisons.
  - suppose three elements $< a_1, a_2, a_3>$ with instance $<6,8,5>$.

Example: insertion sort (n=3)



The Decision Tree Model



Internal node i:j indicates comparison between a_i and a_j.

Leaf node $<\pi(1), \pi(2), \pi(3)>$ indicates ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq a_{\pi(3)}$.

Path of bold lines indicates sorting path for <6,8,5>.

There are total 3!=6 possible permutations (paths).

Summary

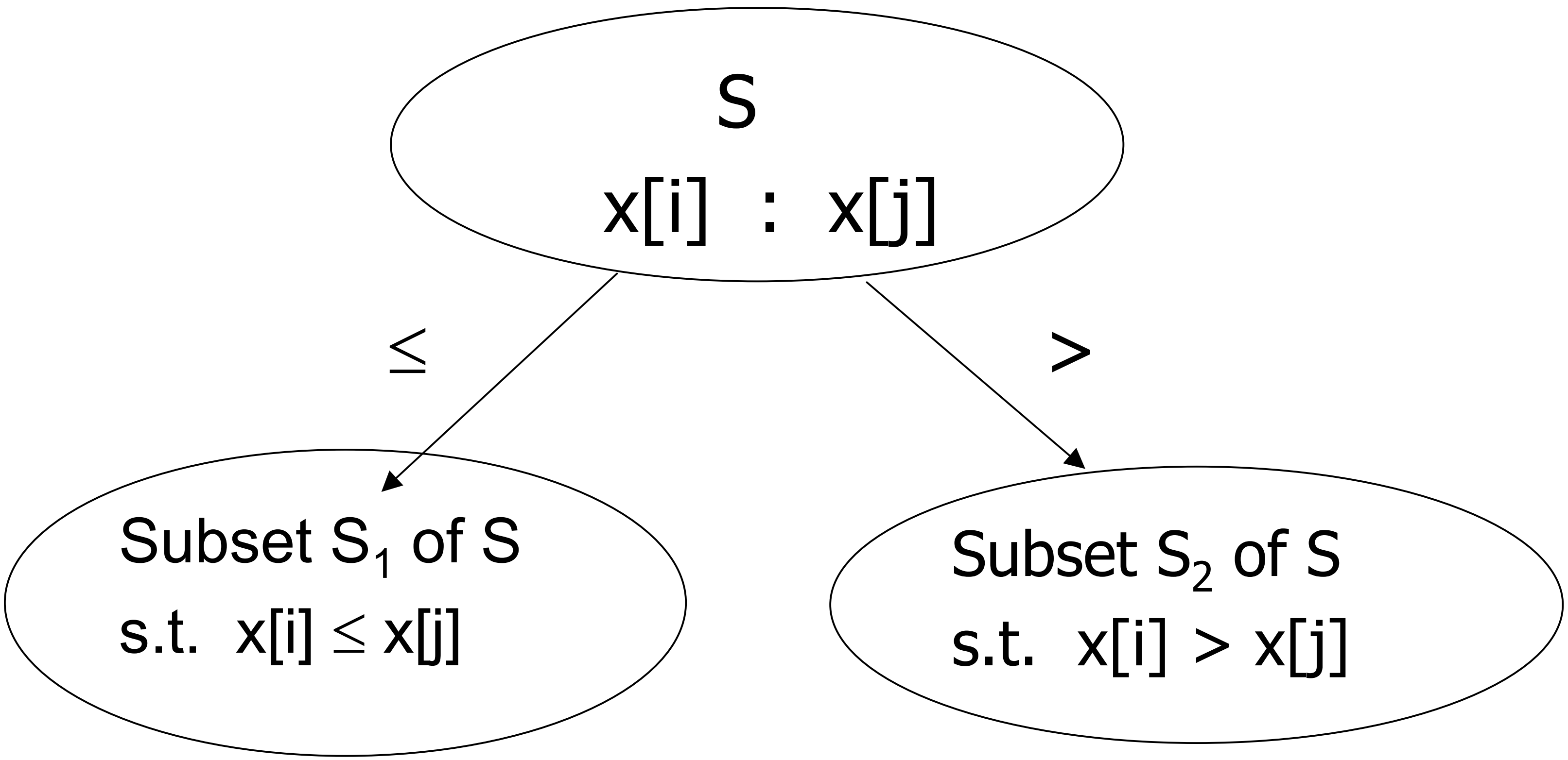
- Only consider comparisons
- Each internal node = 1 comparison
- Start at root, make the first comparison

  - if the outcome is $\leq$ take the LEFT branch

  - if the outcome is > - take the RIGHT branch
- Repeat at each internal node
- Each LEAF represents ONE correct ordering

Intuitive idea

S is a set of permutations



Lower bound for the worst case

- Claim: The decision tree must have at least n! leaves. WHY?
- worst case number of comparisons= the height of the decision tree.
- Claim: Any comparison sort in the worst case needs Ω(n log n) comparisons.
- Suppose height of a decision tree is h, number of paths (i,e,, permutations) is n!.
- Since a binary tree of height h has at most 2^h leaves,

$$n! \leq 2^h \text{, so } h \geq \lg(n!) \geq \Omega(n \lg n)$$

Lower bounds: check your understanding

Can you prove that any algorithm that searches for an element in a sorted array of size n must have running time $\Omega(\lg n)$?

Minimum and Maximum

Problem: Find the maximum and the minimum of n elements.

- Naïve algorithm 1: Find the minimum, then find the maximum -- 2(n-1) comparisons.
- Naïve algorithm 2: Find the minimum, then find the maximum of n-1 elements -- (n-1) + (n-2) = 2n -3 comparisons.

Minimum and Maximum – better algorithms

Problem: Find the maximum and the minimum of n elements.

Approach 1

•Sort n/2 pairs. Find min of losers, max of winners.

comparisons: $n/2 + n/2 - 1 + n/2 - 1 = 3n/2 - 2$.

Is this the best possible?

Approach 2

•Divide into n/2 pairs. Compare the first pair, set winner to current max, loser to current min.

•Sort next pair, compare winner to current max, loser to current min.

#comparisons: $1 + 3(n/2 - 1) = 3n/2 - 2$.

Lower bounds for the MIN and MAX

Claim: Every comparison-based algorithm for finding both the minimum and the maximum of n elements requires at least (3n/2)-2 comparisons.

Idea: Use similar argument as for the minimum

Max = maximum and Min=minimum only if:

Every element other than min has won at least 1

Every element other than max has lost at least 1

A proof?

“Proof” from the web: For each comparison, $x<y$, score a point if this is first comparison that x loses or if y wins and 2 points if both occur. Before the algorithm can terminate n-2 must both win and lose (since they aren't min or max) and 2 elements must either win or lose. Thus, $2(n-2)+2$ points are scored before termination.

Define A to be the set of elements that have not won or lost a comparison. All comparisons between elements in A must score 2 points. All other comparisons can score at most 1 point. Let X be A-A comparisons. Let Y be number of other comparisons. We want to minimize $X+Y$ such that $2X+Y \geq 2n-2$ & $X \leq n/2$ (assume n is even). Given the constraints we want to make X as big as possible. So set $X=n/2$. Then $Y \geq 2n-2-2X \Rightarrow Y \geq 2n-2-n \Rightarrow Y \geq n-2 \Rightarrow X+Y \geq n/2 + n - 2$.

Is the previous proof correct?

Lower bounds for the MIN and MAX

Idea: Define 4 sets: U: has not participated in a comparison

W: has won all comparisons

L: has lost all comparisons

N: has won and lost at least one comparison

Note: All these sets are disjoint.

1. Initially all elements in U.
2. Finally no elements in U, 1 each in W,L and n-2 in N.
3. Each element in N comes from U via W or L.

Lower bounds for the MIN and MAX - contd

Idea: Score a point when an element enters W or L or N for the first time.

Question: Can we ensure that only U-U comparisons result in two points being scored?

Answer: YES! The adversary argument!

The adversary constructs a worst-case input by revealing as little as possible about the inputs.

Lower bounds for the MIN and MAX - contd

Adversary strategy:

U-U: any

U-W: make element of W winner

U-L: make element of L loser

U-N: any

W-W: any (be consistent with before)

W-L/N: make element of W winner

L-L: any (be consistent with before)

L-N: make element of L loser

Lower bounds for the MIN and MAX – contd.

We need to score 2n–2 points. At most n/2 U-U comparisons can be made – gives n points.

To move n-2 elements to N, we need another n-2 comparisons.

Next: Linear sorting

Q: Can we beat the Ω(n log n) lower bound for sorting?

A: In general no, but in some special cases YES!

Ch 7: Sorting in linear time