Sorting (Part II)

CSE 373

Data Structures

How fast can we sort?

- Heapsort, Mergesort, and Quicksort all run in O(N log N) best case running time
- Can we do any better?
- No, if sorting is comparison-based.

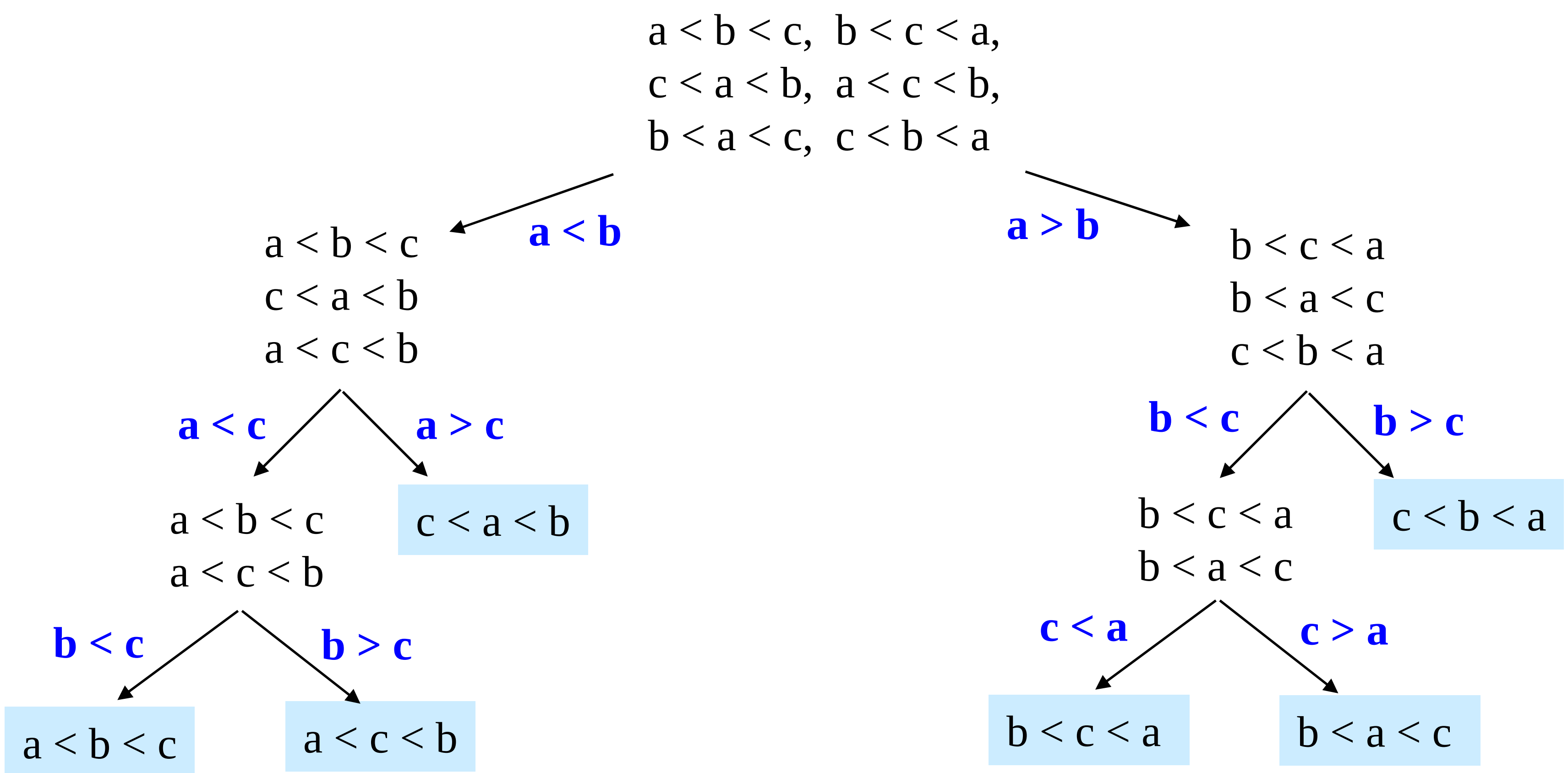
Sorting Model

- Basic assumption: we can only compare two elements at a time
  - we can only reduce the possible solution space by half each time we make a comparison
- Suppose you are given N elements
  - Assume no duplicates
- How many possible orderings can you get?
  - Example: a, b, c (N = 3)

Permutations

- How many possible orderings can you get?
  - Example: a, b, c (N = 3)
  - (a b c), (a c b), (b a c), (b c a), (c a b), (c b a)
  - 6 orderings = 3•2•1 = 3! (i.e., “3 factorial”)
  - All the possible permutations of a set of 3 elements
- For N elements
  - N choices for the first position, (N-1) choices for the second position, …, (2) choices, 1 choice
  - $N(N-1)(N-2)\cdots(2)(1)= N!$ possible orderings

Decision Tree



The leaves contain all the possible orderings of a, b, c

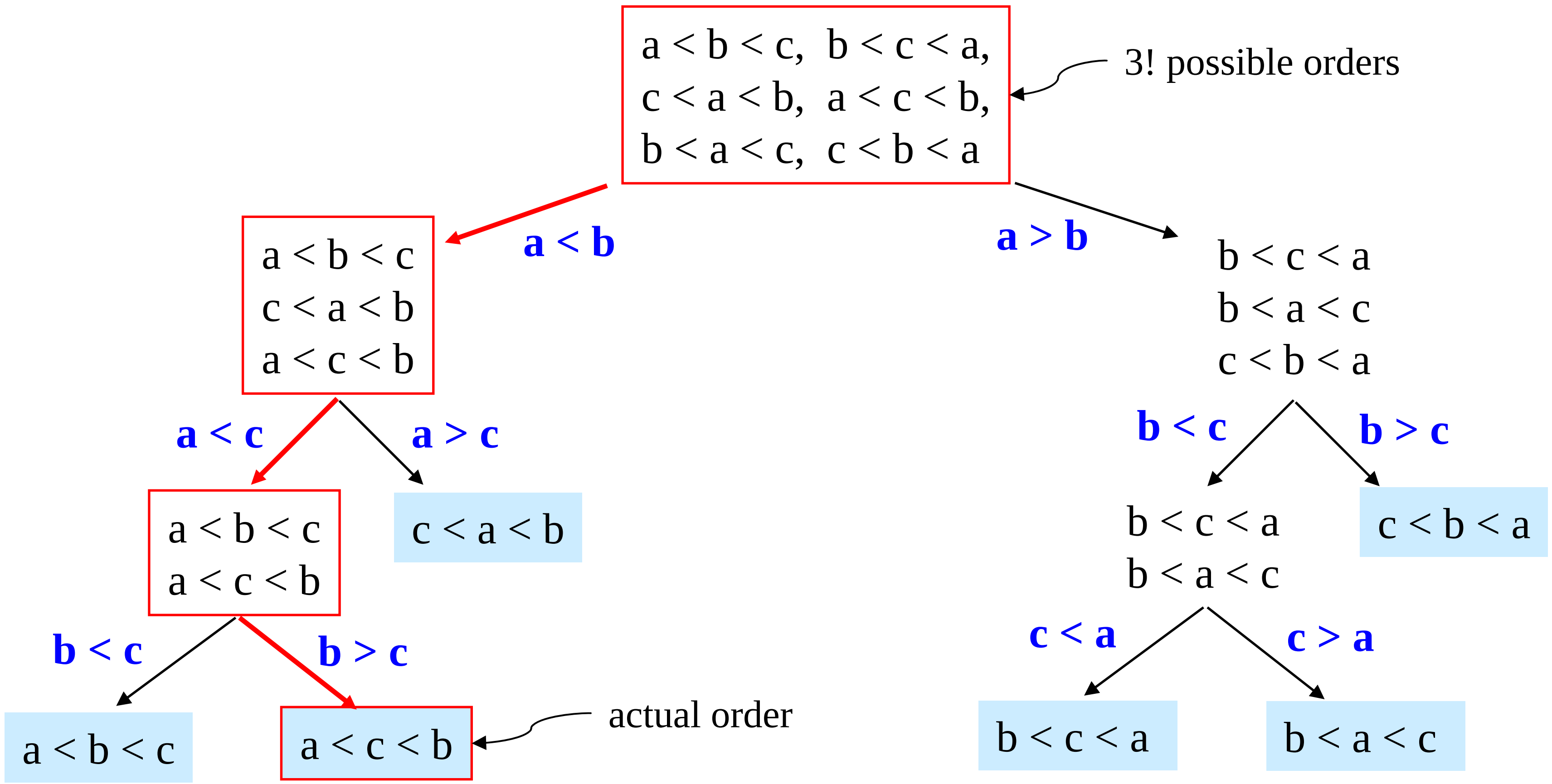
Decision Trees

- A Decision Tree is a Binary Tree such that:
  - Each node = a set of orderings
    - i.e., the remaining solution space
  - Each edge = 1 comparison
  - Each leaf = 1 unique ordering
  - How many leaves for N distinct elements?
    - N!, i.e., a leaf for each possible ordering
- Only 1 leaf has the ordering that is the desired correctly sorted arrangement

Decision Trees and Sorting

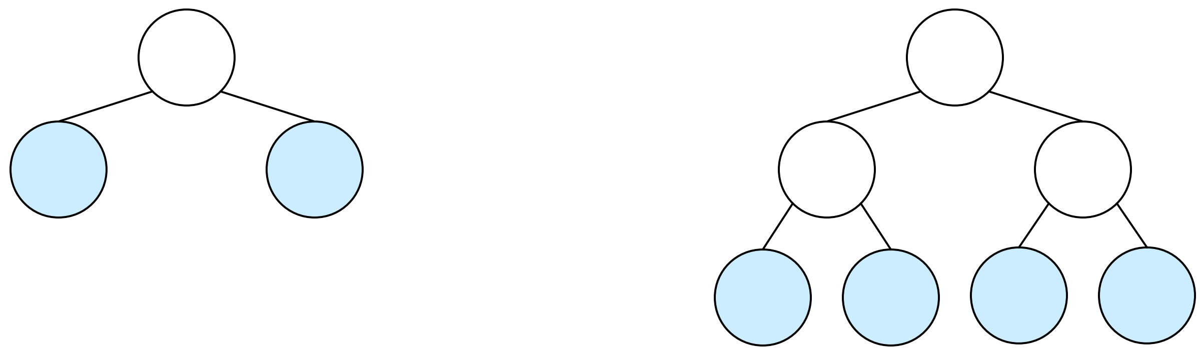
- Every comparison-based sorting algorithm corresponds to a decision tree
  - › Finds correct leaf by choosing edges to follow
    - i.e., by making comparisons
  - › Each decision reduces the possible solution space by one half
- Run time is $\geq$ maximum no. of comparisons
  - › maximum number of comparisons is the length of the longest path in the decision tree, i.e. the height of the tree

Decision Tree Example



How many leaves on a tree?

- Suppose you have a binary tree of height d . How many leaves can the tree have?
  - › d = 1 → at most 2 leaves,
  - › d = 2 → at most 4 leaves, etc.



Lower bound on Height

- A binary tree of height d has at most $\mathbf{2^d}$ leaves
  - depth $d = 1 \rightarrow 2$ leaves, $d = 2 \rightarrow 4$ leaves, etc.
  - Can prove by induction
- Number of leaves, $L \leq 2^d$
- Height $d \geq \log_2 L$
- The decision tree has N! leaves
- So the decision tree has height $d \geq \log_2(N!)$

Upper Bounds and Lower Bounds

- f(n) is O(g(n)) means that f(n) does not grow any faster than g(n)
  - › g(n) is an upper bound for f(n)
- f(n) is Ω(g(n)) means that f(n) grows ar least as fast as g(n)
  - › g(n) is a lower bound for f(n)
  - › f(n) is Ω(g(n)) if g(n) is O(f(n))

$\log(N!)$ is $\Omega(N \log N)$

$$\log(N!) = \log\big(N \cdot (N-1) \cdot (N-2) \cdots (2) \cdot (1)\big)$$

$$= \log N + \log(N-1) + \log(N-2) + \cdots + \log 2 + \log 1$$

select just the first N/2 terms

$$\geq \log N + \log(N-1) + \log(N-2) + \cdots + \log \frac{N}{2}$$

each of the selected terms is ≥ logN/2

$$\geq \frac{N}{2} \log \frac{N}{2}$$

$$\geq \frac{N}{2}(\log N - \log 2) = \frac{N}{2} \log N - \frac{N}{2}$$

$$= \Omega\big(N \log N\big)$$

$$n! \approx \sqrt{2\pi n}\,(n/e)^n$$

Sterling's formula

$\Omega(N \log N)$

- Run time of any comparison-based sorting algorithm is **$\Omega(N \log N)$**
- Can we do better if we don't use comparisons?

Bucket Sort

- *n* Keys to sort in range [0,N-1]
- Have N buckets: bucket *i* will contain the elements with key value *i*
- Pass 1: place elements in their respective buckets: O(*n*)
- Pass 2: concatenate the N buckets: O(*n*+N) since have to check empty buckets
- Needs extra space
- Good only if N not too large compared to *n*

Radix Sort: Sorting integers

- Historically goes back to the 1890 census.
- Radix sort = multi-pass bucket sort of integers in the range 0 to B^P-1
- Bucket-sort from least significant to most significant “digit” (base B)
- Requires P(B+N) operations where P is the number of passes (the number of base B digits in the largest possible input number).
- If P and B are constants then O(N) time to sort!

Radix Sort Example

Input data

478
537
9
721
3
38
123
67

Bucket sort
by 1's digit

0	1	2	3	4	5	6	7	8	9
	721		3 123				537 67	478 38	9

After 1st pass

721
3
123
537
67
478
38
9

This example uses decimal digits for simplicity of demonstration. Larger bucket counts should be used in an actual implementation.

Radix Sort Example

After 1st pass

721
3
123
537
67
478
38
9

Bucket sort
by 10's
digit

0	1	2	3	4	5	6	7	8	9
03 09		721 123	537 38			67	478		

After 2nd pass

3
9
721
123
537
38
67
478

Radix Sort Example

After 2nd pass

3
9
721
123
537
38
67
478

Bucket sort by 100's digit

0	1	2	3	4	5	6	7	8	9
003 009 038 067	123			478	537		721		

After 3rd pass

3
9
38
67
123
478
537
721

Invariant: after k passes the low order k digits are sorted.

Implementation Options

- Linked List
  - Linked List of data, bucket array of linked lists.
  - Concatenate lists for each pass.
- Array / Linked List
  - Array of data, bucket array of linked lists.
- Array / Array
  - Array of data, array for all buckets.
  - Requires counting.

Array / Array

	Data Array		Count Array		Address Array		Target Array	
0	478	0	0	0	0	0	721	1
1	537	1	1	1	0	1	3	3
2	9	2	0	2	1	2	123	
3	721	3	2	3	1	3	537	7
4	3	4	0	4	3	4	67	
5	38	5	0	5	3	5	478	8
6	123	6	0	6	3	6	38	
7	67	7	2	7	3	7	9	9
		8	2	8	5			
		9	1	9	7			

Bucket i ranges from add[i] to add[i+1]-1

add[0] := 0

add[i] := add[i-1] + count[i-1], i > 0

Array / Array

- Pass 1 (over A)
  - Calculate counts and addresses for 1st "digit"
- Pass 2 (over T)
  - Move data from A to T
  - Calculate counts and addresses for 2nd "digit"
- Pass 3 (over A)
  - Move data from T to A
  - Calculate counts and addresses for 3nd "digit"
- …
- In the end an additional copy may be needed.

Choosing Parameters for Radix Sort

- N number of integers – given
- m bit numbers - given
- B number of buckets
  - › $B = 2^r$: power of 2 so that calculations can be done by shifting.
  - › N/B not too small, otherwise too many empty buckets.
  - › P = m/r should be small.
- Example – 1 million 64 bit numbers. Choose $B = 2^{16}$ =65,536. 1 Million / B$\approx$ 15 numbers per bucket. P = 64/16 = 4 passes.

Properties of Radix Sort

- Not in-place
  - needs lots of auxiliary storage.
- Stable
  - equal keys always end up in same bucket in the same order.
- Fast
  - B = 2^r buckets on m bit numbers

$$O\left(\frac{m}{r}\left(n+2^r\right)\right) \text{ time}$$

Internal versus External Sorting

- So far assumed that accessing A[i] is fast – Array A is stored in internal memory
  - Algorithms so far are good for internal sorting
- What if A is so large that it doesn't fit in internal memory?
  - Data on disk
  - Delay in accessing A[i] –need to get many records (keys) at a time

Internal versus External Sorting

- Need sorting algorithms that minimize disk access time
  - › External sorting – Basic Idea:
    - Load chunk of data into main memory, sort, store this “run” on disk
    - Use the Merge routine from Mergesort to merge runs
    - Repeat until you have only one run (one sorted chunk)

Summary of Sorting

- Sorting choices:
  - O(N^2) – Insertion Sort
  - O(N log N) average case running time:
    - Heapsort: In-place, not stable.
    - Mergesort: O(N) extra space, stable.
    - Quicksort: claimed fastest in practice but, O(N^2) worst case. Needs extra storage for recursion. Not stable.
  - O(N) – Radix Sort: fast and stable. Not comparison based. Not in-place.