Lower Bounds

- So far we have concentrated on the question: *Given a problem* Π *can we construct an algorithm that solves* Π *in* $O(f(n))$ *time*? In other words, is there an algorithm A that satisfies:
$$T_A(n) = \max_{|X|=n} T_A(X) \in O(f(n))$$
- *Goal*: find $f(n)$ that grows as *slowly* as possible
- Today we concentrate on proving statements of the form: *any* algorithm that solves *X must* take $\Omega(g(n))$ time
$$T_\Pi(n) = \min_{A \text{ solves } \Pi} T_A(n) = \min_{A \text{ solves } \Pi} \max_{|X|=n} T_A(X) \in \Omega(g(n))$$
  - The complexity $T_\Pi(n)$ of a problem Π is the complexity of the *best algorithm* that solves Π
  - *New Goal*: find $g(n)$ that grows as *fast* as possible

Lower Bounds…

- The ratio of the slowest growing upper bound to the fastest growing lower bound, $f(n)/g(n)$, is a kind of "gap" for Π

  Example: If Π has a lower bound of $\Omega(n \log n)$ and the best known algorithm solves Π in $O(n \log^2 n)$ time then there is a gap of $\log n$ for Π
- When A satisfies $f(n) \in \Theta(g(n))$, A is *optimal*
- Improving lower bounds is considerably harder than improving upper bounds, because a lower bound applies to *all* algorithms that solve Π
- *Problem*: what do we mean by '*all algorithms*'?

Models of Computation

- We first specify the ***model of computation***, i.e., the kinds of algorithms allowed and the cost of the model operations
- Lower bounds apply to a specific model of computation
- Model today: ***decision tree***, a k-ary tree such that:
  - Each internal node is labeled with a query about the input
  - Edges out of a node correspond to answer to the query
  - Each leaf is labeled with a possible output
  - A specific computation is a path from root to a leaf, where the answers tell us what to compute next
  - Cost is the number of queries asked
- Each query has $\leq k$ branches, N possible total outputs $\Rightarrow$ $\geq \lceil \log_k N \rceil$ queries are needed in the worst case

3

Lower Bounds with Decision Trees

- Lower bounds to many problems can be obtained using decision trees
  - Searching a sorted list
  - Searching an unsorted list
  - Finding the i-th smallest element of a list
  - Finding the mode of a list
  - Sorting a list
  - Merging two sorted lists
  - Find the elements common to two lists
  - Determine if all elements of a list are distinct
  - Determine if two lists have the same elements

4

Comparison Trees

- A *comparison tree* is a type of decision tree where each query involves the comparison of two values
  - Each internal node v is labeled with a comparison $x: y$ for some input keys x and y
  - Each internal node has 2 or 3 outcomes
  - Each leaf is labeled with an output of Π on some input of size n
  - For each input x, there is a path(x) from root to a leaf such that every edge (u, v) in path(x) is labeled with the comparison performed at u
  - Tree is correct if for every x, leaf in path(x) is a valid output

5

Comparison Trees…

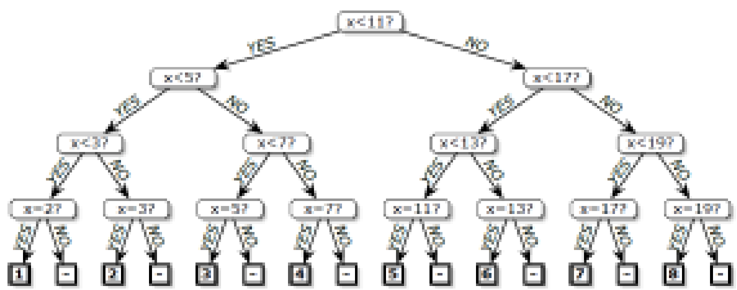
- You have one tree for each combination of input size and algorithm for problem Π
- The number of leaves in a decision tree of order n for Π is greater than or equal to the number of distinct outputs of Π on inputs of size n
- A k-ary tree with L leaves has height at least $\lceil \log_k L \rceil$, a lower bound for problem Π

6

Example: Searching a Sorted List

- Each searching strategy and input size has a tree
- Below is the tree for standard *binary search*

2	**3**	**5**	**7**	**11**	**13**	**17**	**19**



- Depth is in $\Omega(\log n)$Independent of tree organization

7

Exercise

1. Describe the decision tree for the following search strategies:
   - Linear search
   - Jump search
   - Exponential search
2. Use comparison trees to derive a lower bound on the complexity of merging two sorted lists of size n each

8

Lower Bounds for Sorting

- How fast can we sort?
  – Answer depends on computational model
- So far:
  – Insertion sort takes $\Theta(n^2)$ (worst case)
  – Quicksort takes $\Theta(n \log n)$ (expected)
  – Merge sort takes $\Theta(n \log n)$ (worst case)
  – Heapsort takes $\Theta(n \log n)$ (worst case)
- Can we do better than $\Theta(n \log n)$?
- These algorithms share the same model
  – Will provide a lower bound for this model and then beat it (for *restricted* inputs) by changing the model

Comparison Sort Model

- Uses the comparison tree model, i.e., the basic operation is the *comparison of two elements*
  – Only use comparisons to determine relative order (resulting algorithm is called a *comparison sort*
  – Only count comparisons to determine complexity
- Lower bounds
  – $\Omega(n)$ to examine all the input
  – All sorts seen so far are comparison sorts and take $\Omega(n \log n)$ in the worst case
  – Will show $\Omega(n \log n)$ lower bound for this model

2

Comparison Trees for Sorting

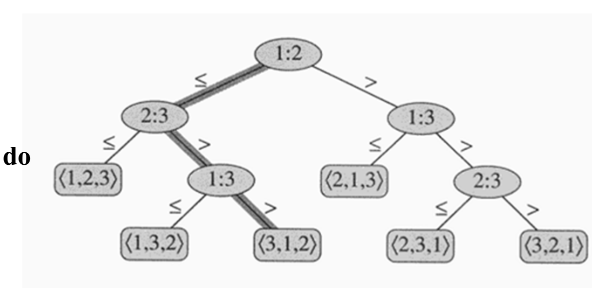
- A *comparison tree* is used as an abstraction of a comparison sort
- The tree represents the set of *all* possible comparisons made by a <u>fixed</u> <u>algorithm</u> on inputs of a <u>fixed</u> <u>size</u>
- Abstracts away everything else, such as control and data movement
- Only comparisons are counted

11

Example

- Insertion sort, $n = 3$, e.g., sort $\langle 3,5,2 \rangle$

```
Sort(A, n)
for j ← 2 to n do
    k ← A[j]
    i ← j − 1
    while i > 0 and A[i] > k do
        A[i+1] ← A[i]
        i ← i + 1
    A[i+1] ← k
```



- Each node labeled with *original* element indices
- Each leaf labeled by permutation found by algorithm
- Path in bold corresponds to $a_3 \leq a_1 \leq a_2$

12

More generally…

- Want to sort $\langle a_1,\ldots,a_n\rangle$
- Each internal node has label a_i:a_j, where $i,j\in\{1,\ldots n\}$
- Left subtree contains comparisons performed *after* determining that $a_i \le a_j$
- Right subtree contains subsequent comparisons for the case $a_i > a_j$
- Each leaf node has a permutation of $\langle 1,\ldots,n\rangle$ that corresponds to the correct sorted order of the input

13

Properties

For a particular (deterministic) algorithm

- One tree for each n
- All possible execution traces are represented

- A specific run $\Rightarrow$ a path from root to leaf
- How many leaves does a decision tree have?
- What is the length of longest path from root to leaf?

Depends on the algorithm!

- insertion sort?
- heapsort?
- merge sort?
- quicksort?

14

Lower Bound for Sorting

Theorem. Any decision tree for sorting n elements has height $\Omega(n \log n)$

Proof.

A binary tree of height h has $\leq 2^h$ leaves

Decision tree for correct algorithm has $\geq n!$ leaves

$\Rightarrow n! \leq \#\text{leaves} \leq 2^h \Rightarrow h \geq \log n!$

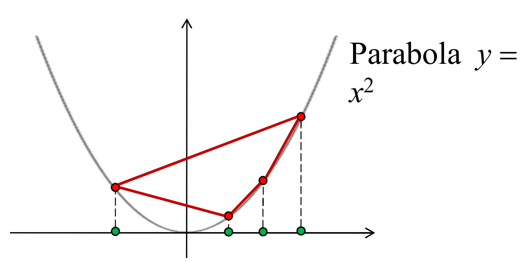
Corollary. Heapsort and Merge sort are asymptotically optimal (under the comparison model of sorting)

15

Other Lower Bounds

- Given a problem X, one can often use a **reduction** from *Sorting* to X to show that X has a $\Omega(n \log n)$ lower bound as well
- The reduction must require $o(n \log n)$ time

Example: Sorting $\leq_{O(n)}$ Convex-Hull



Example: Convex Hull $\leq_{O(n)}$ Triangulation

16

Digit-Based Sorting

- So far, when sorting, we have viewed the input keys as abstract objects that can only be examined via comparisons
- Now, we view each input key as a sequence of "digits"
- Digits can be individually manipulated
- New point of view leads to:
  - Fast sorting algorithm (radix sort)
  - Online data structure (tries)

17

Sorting in Linear Time

- Non-comparison sorts
- Need additional assumptions about items to be sorted

Example: Counting Sort

Assumption: input integers in $\{0, 1, 2, ..., k\}$

- Input: $A[1..n]$ Output: $B[1..n]$ Auxiliary: $C[0..k]$
- Idea: for each i in $1..n$ compute rank of $A[i]$
- $C[i]$ = rank of $A[i]$ = # elements from A that are $\leq i$

18

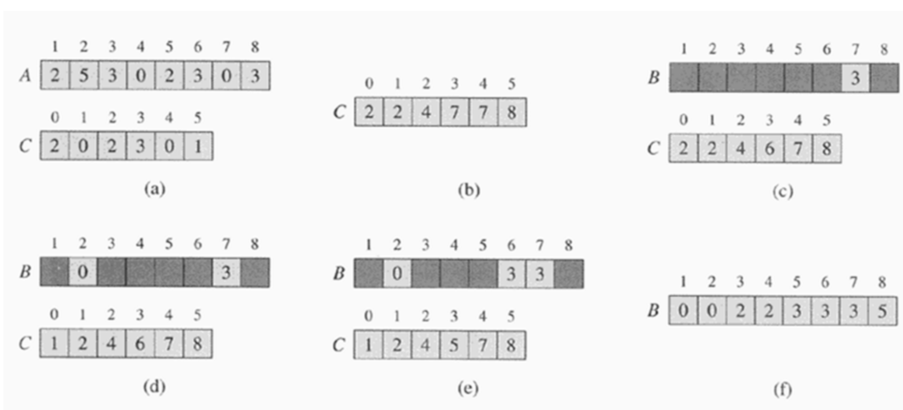
```
COUNTING-SORT(A, B, n, k)
for i ← 0 to k
     do C[i] ← 0
for j ← 1 to n
     do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
     do C[i] ← C[i] + C[i − 1]
for j ← n downto 1
     do B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] − 1
```

Analysis.

- Running time: $\Theta(n + k)$ which is $\Theta(n)$ if $k = O(n)$
- How big a k is practical?
  32-bit numbers? 16-bit? 8-bit?

19

Example



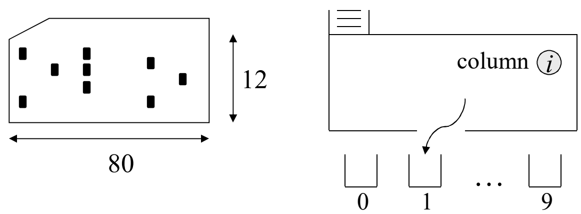
20

What if k is big?

- Assume each input number a has d "digits"

  $a = a_d a_{d-1} \ldots a_2 a_1$

- IBM's card sorting machine



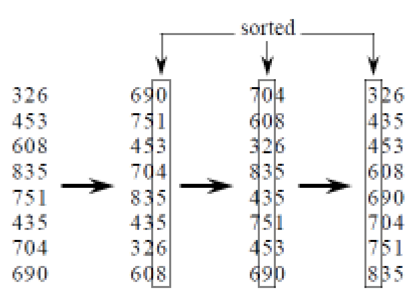
- Two strategies based on sorting one column at a time
  - most significant digit first
  - least significant digit first

21

Radix Sort

Input value x consists of d digits: $x = x_d x_{d-1} \cdots x_2 x_1$

```
RADIX-SORT(A, d)
for i ← 1 to d
    do use a stable sort to sort array A on digit i
```



22

Analysis

- Correctness
  - Prove by induction on d (number of passes)
    - Assume input already sorted on digits 1,..., $i - 1$
    - Argue that sort on digit i, leaves digits 1,..., i sorted
- Time complexity:
  - $O(d\,(n + k))$ when coupled with counting sort
  - depends on
    - stable sort used
    - d, which depends on n and number base used

23

How do your break keys into digits?

- Each key has b bits
- Each digit is r bits $\Rightarrow d = \lceil b/r \rceil$
- With counting sort, $k = 2^r - 1$

  Example: b=32, r=8 $\Rightarrow d$=32/8=4, k=255
- Time: $\Theta(\, b/r\,(n + 2^r)\,)$
- How do you choose r?

  $r \approx \log n \Rightarrow$ time is $\Theta(\, bn/\log n\,)$

  $r < \log n \Rightarrow b/r > b/\log n$ but $(n+2^r) = \Omega(n)$

  $r > \log n \Rightarrow (n+2^r)$ gets big quickly

24

Merge Sort or Radix Sort?

- Sort 1 million 32-bit integers
- Merge sort performs 20 "passes"
- Since $\lceil \log 10^6 \rceil = 20$, radix sort performs $\lceil 32/20 \rceil = 2$ calls to counting sort
- Each call to counting sort requires 4 passes

25

Bucket Sort

- Input: array $A[1..n]$ of numbers in $[0,1)$
  Uses auxiliary array $B[0..n-1]$ of linked lists
- Algorithm
  - Divide $[0,1)$ into n equal size buckets
  - Place each input value into corresponding bucket
  - Sort the buckets independently
  - Concatenate
- Works well if input is uniformly distributed

26

BUCKET-SORT(A, n)

for $i \leftarrow 1$ **to** n

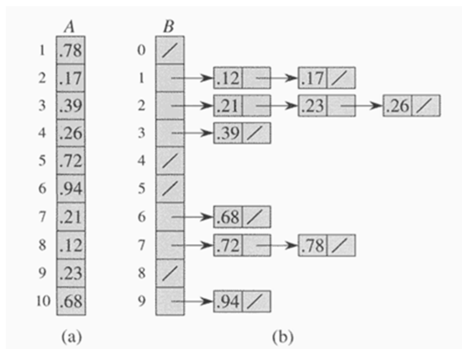
  do insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$

for $i \leftarrow 0$ **to** $n - 1$

  do sort list $B[i]$ with insertion sort

concatenate lists $B[0], B[1], \ldots, B[n-1]$ together in order

return the concatenated lists



Correctness

Consider to arbitrary keys: $A[i]$ and $A[j]$

$A[i] < A[j] \Rightarrow \lfloor n{\cdot}A[i] \rfloor \leq \lfloor n{\cdot}A[j] \rfloor$

$\Rightarrow A[i]$ is placed on same bucket as $A[j]$ **or** in bucket with lower index

If same bucket, then insertion sort fixes the order

If different bucket, concatenation fixes the order

Probabilistic Analysis

- Assume input generated by random process
- Let n_i denote size of $B[i]$ (a random variable)

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

Claim. $E[(n_i)^2] = 2 - 1/n$

29

Proof

$$X_{ij} = I\{A[j] \text{ falls in bucket } i\}$$

$$n_i = \sum_{j=1}^{n} X_{ij}$$

$$E[n_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} E\left[X_{ij}^2\right] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} E\left[X_{ij}X_{ik}\right]$$

30

Key observations

- X_{ij}^2 is a decision variable, same as X_{ij}
- X_{ij} and X_{ik} are independent random variables

$$\sum_{j=1}^{n} E\left[X_{ij}^2\right] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} E\left[X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} E\left[X_{ij}^2\right] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} E\left[X_{ij}\right]E\left[X_{ik}\right]$$

$$= \sum_{j=1}^{n} \frac{1}{n} + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \frac{1}{n}\cdot\frac{1}{n} = 1 + 2\binom{n}{2}\frac{1}{n^2}$$

$$= 1 + 2\,\frac{n(n-1)}{2}\cdot\frac{1}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n} \quad \blacksquare$$

31