



National Cheng Kung University

A large, lush green tree with a thick trunk and dense foliage, set against a dark background.

Chapter 8

Sorting in Linear Time

Sun-Yuan Hsieh

謝孫源 教授

成功大學資訊工程學系



Lower bounds for sorting

Lower bounds

- ▶ $\Omega(n)$ to examine all the input.
- ▶ All sorts seen so far are $\Omega(n \lg n)$.
- ▶ We'll show that $\Omega(n \lg n)$ is a lower bound for comparison sorts.

Decision tree

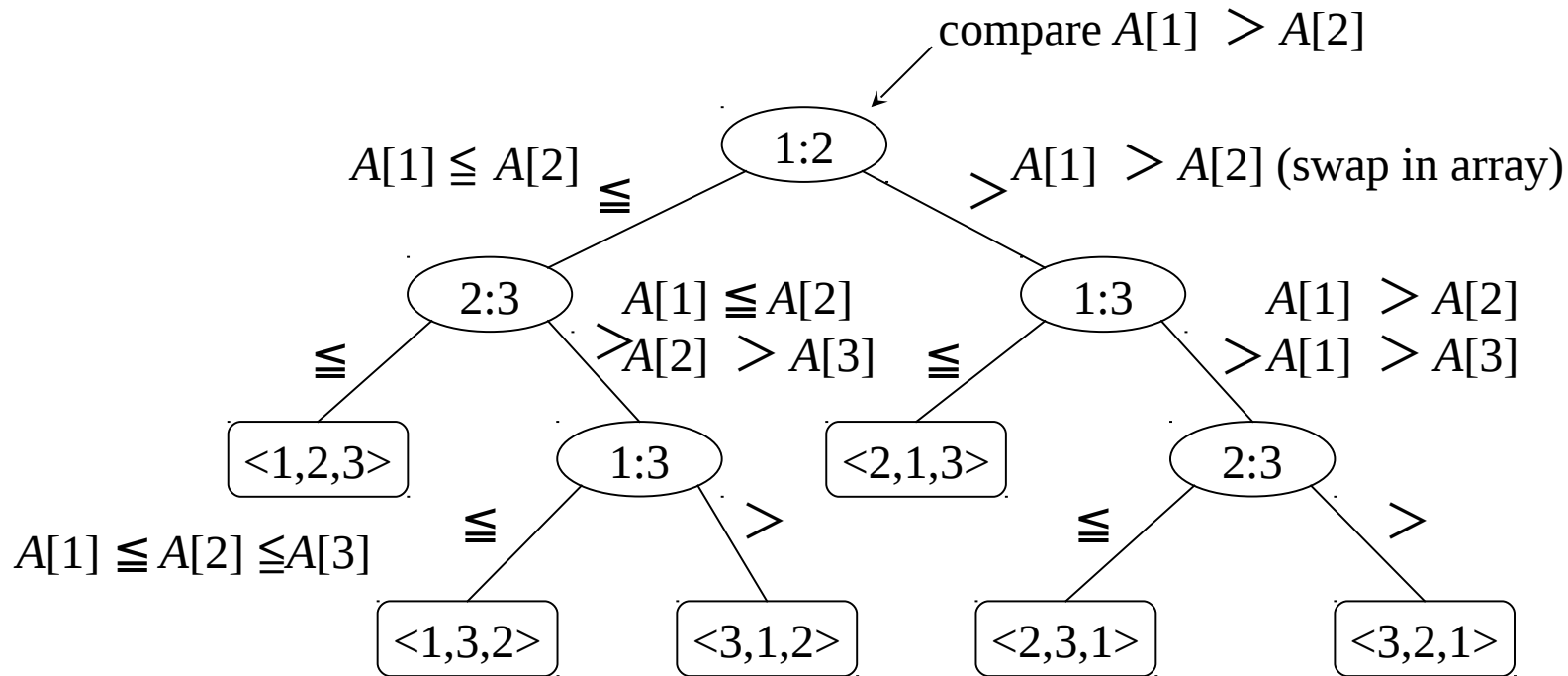
- ▶ Abstraction of any comparison sort.
- ▶ Represents comparisons made by
 - ▷ a specific sorting algorithm
 - ▷ on inputs of a given size.
- ▶ Abstracts away everything else: control and data movement.
- ▶ We're counting *only* comparisons.

For insertion sort on 3 elements:



成功大學

COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY



[Each internal node is labeled by indices of array elements **from their original positions**. Each leaf is labeled by the permutation of orders that the algorithm determines.]



How many leaves on the decision tree? There are $\geq n!$ leaves, because every permutation appears at least once.

For any comparison sort,

- ▶ 1 tree for each n .
- ▶ View the tree as if the algorithm splits in two at each node, based on the information it has determined up to that point.
- ▶ The tree models all possible execution traces.

What is the length of the longest path from root to leaf ?



成功大學

COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY



- ▶ Depends on the algorithm
- ▶ Insertion sort: $\Theta(n^2)$
- ▶ Merge sort: $\Theta(n \lg n)$



Lemma

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

In other words:

- ▶ $l = \#$ of leaves,
- ▶ $h = \text{height}$,
- ▶ Then $l \leq 2^h$.



Proof: By induction on h .

Basis: $h = 0$. Tree is just one node, which is a leaf. $2^h = 1$.

Inductive step: Assume true for height $= h - 1$.

Extend tree of height $h - 1$ by making as many new leaves as possible.
Each leaf becomes parent to two new leaves.

$$\begin{aligned}\# \text{ of leaves for height } h &= 2 \cdot (\# \text{ of leaves for height } h - 1) \\ &\leq 2 \cdot 2^{h-1} && \text{(ind. hypothesis)} \\ &= 2^h. && \text{(lemma)}\end{aligned}$$

Theorem. Any decision tree that sorts n elements has height $\Omega(n \lg n)$



成功大學

COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY



Proof

- ▶ $l \geq n!$
- ▶ By lemma, $n! \leq l \leq 2^h$ or $2^h \geq n!$
- ▶ Take logs: $h \geq \lg(n!)$
- ▶ Use Stirling's approximation: $n! > (n/e)^n$ (by equation (3.16))

$$h \geq \lg(n/e)^n$$

$$= n \lg(n/e)$$

$$= n \lg n - n \lg e$$

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

(theorem)



Corollary

Heapsort and merge sort are asymptotically optimal comparison sort.



Sorting in linear time

Non-comparison sorts.

Counting sort

Depends on a *key assumption*: numbers to be sorted are integers in $\{0, 1, \dots, k\}$.

Input: $A[1..n]$, where $A[j] \in \{0, 1, \dots, k\}$ for $j = 1, 2, \dots, n$.
Array A and values n and k are given as parameters.

Output: $B[1..n]$, sorted. B is assumed to be already allocated and is given as a parameter.

Auxiliary storage: $C[0..k]$

COUNTING-SORT(A, B, n, k)

for $i \leftarrow 0$ **to** k

do $C[i] \leftarrow 0$

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

do $C[A[j]] \leftarrow C[A[j]] + 1$

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

do $C[i] \leftarrow C[i] + C[i - 1]$

for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$



Do an example for $A = 2_1, 5_1, 3_1, 0_1, 2_2, 3_2, 0_2, 3_3$

Counting sort is ***stable*** (keys with same value appear in same order in output as they did in input) because of how the last loop works.



The operation of Counting-sort on an input array $A[1..8]$



COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3
	0	1	2	3	4	5		
C	2	0	2	3	0	1		

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

	1	2	3	4	5	6	7	8
B							3	
	0	1	2	3	4	5		
C	2	2	4	6	7	8		

(c)

	1	2	3	4	5	6	7	8
B		0					3	
	0	1	2	3	4	5		
C	1	2	4	6	7	8		

(d)

	1	2	3	4	5	6	7	8
B		0				3	3	
	0	1	2	3	4	5		
C	1	2	4	5	7	8		

(e)

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

(f)

Analysis: $\Theta(n + k)$, which is $\Theta(n)$ if $k = O(n)$.

How big a k is practical?

- ▶ Good for sorting 32-bit values? No.
- ▶ 16-bit? Probably not.
- ▶ 8-bit? Maybe, depending on n .
- ▶ 4-bit? Probably (unless n is really small).

Counting sort will be used in radix sort.

Radix sort



COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY

Key idea: Sort *least* significant digits first.

To sort d digits:

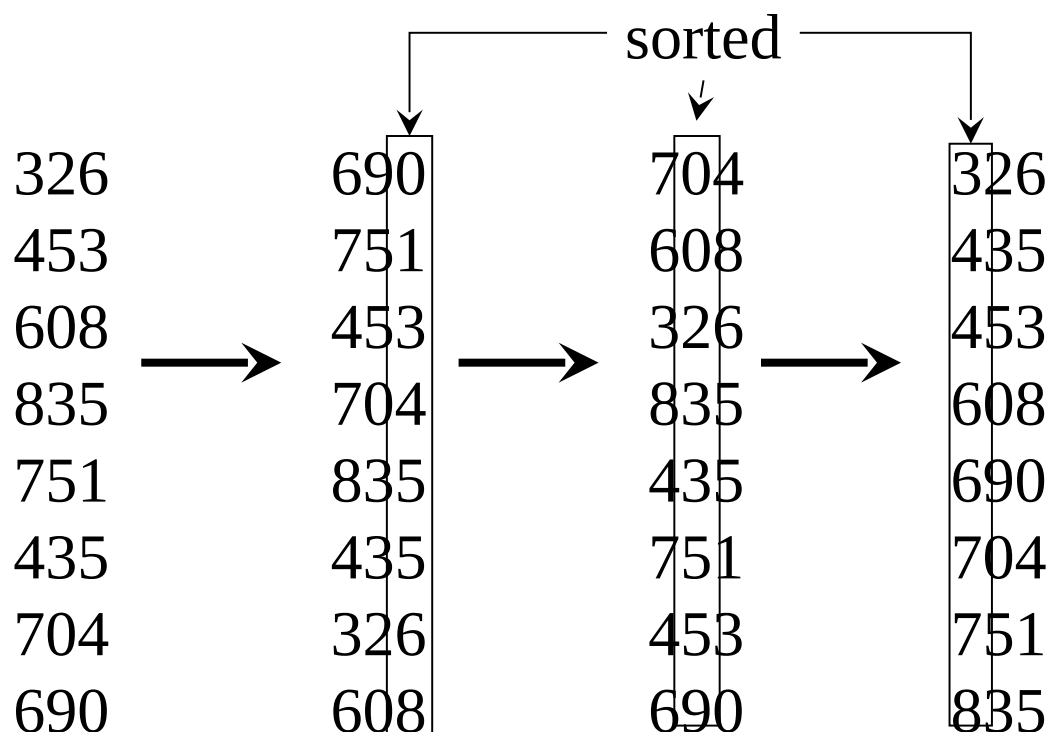
RADIX-SORT(A, d)

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

do use a stable sort to sort array A on digit i



Example:





Correctness:

- ▶ Induction on number of passes (i in pseudocode).
- ▶ Assume digits $1, 2, \dots, i - 1$ are sorted.
- ▶ Show that a stable sort on digit i leaves digits $1, \dots, i$ sorted:
 - ▷ If 2 digits in position i are different, ordering by position i is correct, and positions $1, \dots, i - 1$ are irrelevant.
 - ▷ If 2 digits in position i are equal, numbers are already in the right order (by inductive hypothesis). The stable sort on digit i leaves them in the right order.

This argument shows why it's so important to use a stable sort for intermediate sort.

Analysis: Assume that we use counting sort as the intermediate sort.

- ▶ $\Theta(n + k)$ per pass (digits in range $0, \dots, k$)
- ▶ d passes
- ▶ $\Theta(d(n + k))$ total
- ▶ If $k = O(n)$, time = $\Theta(dn)$.

How to break each key into digits?

- ▶ n words
- ▶ b bits/word
- ▶ Break into r -bit digits. Have $d = \lceil b/r \rceil$
- ▶ Use counting sort, $k = 2^r - 1$

Example: 32-bit words, 8-bit digits. $b=32$, $r=8$, $d = \lceil 32/8 \rceil$
 $=4$, $k=2^8 - 1 = 255$

- ▶ Time = $\Theta\left(\frac{b}{r}(n + 2^r)\right)$



How to choose r ? Balance b/r and $n + 2^r$. Choosing $r \approx \lg n$ gives us

$$\Theta\left(\frac{b}{\lg n}(n + n)\right) = \Theta(bn / \lg n)$$

► If we choose $r < \lg n$, then $b/r > b/\lg n$, and $n + 2^r$ term doesn't improve.

► If we choose $r > \lg n$, then $n + 2^r$ term gets big. Example:

$$r = 2 \lg n \Rightarrow 2^r = 2^{2 \lg n} = (2^{\lg n})^2 = n^2.$$

So, to sort 2^{16} 32-bit numbers, use $r = \lg 2^{16} = 16$ bit. $\lceil b/r \rceil = 2$ passes.

Compare radix sort to merge sort and quicksort:



成功大學

COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY



- ▶ 1 million (2^{20}) 32-bit integers.
- ▶ Radix sort: $\lceil 32/20 \rceil = 2$ passes.
- ▶ Merge sort/quicksort: $\lg n = 20$ passes.
- ▶ Remember, though, that each radix sort “pass” is really 2 passes—one to take census, and one to move data.

How does radix sort violate the ground rules for a comparison sort?

- ▶ Using counting sort allows us to gain information about keys by means other than directly comparing 2 keys.
- ▶ Used keys as array indices.



Bucket sort

Assumes the input is generated by a random process that distributes elements uniformly over $[0, 1)$.

Idea:

- ▶ Divide $[0, 1)$ into n equal-sized *buckets*.
- ▶ Distribute the n input values into the buckets.
- ▶ Sort each bucket.
- ▶ Then go through buckets in order, listing elements in each one.

Input: $A[1..n]$, where $0 \leq A[i] < 1$ for all i .

Auxiliary array: $B[0..n - 1]$ of linked lists, each list initially empty.

BUCKET-SORT(A, n)

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

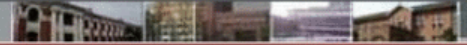
do insert $A[i]$ into list $B \lfloor n \cdot A[i] \rfloor$

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

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

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

return the concatenated lists



Correctness: Consider $A[i]$, $A[j]$. Assume without loss of generality that $A[i] \leq A[j]$. Then $\lfloor n \cdot A[i] \rfloor \leq \lfloor n \cdot A[j] \rfloor$. So $A[i]$ is placed into the same bucket as $A[j]$ or into a bucket with a lower index.

- ▶ If same bucket, insertion sort fixes up.
- ▶ If earlier bucket, concatenation of lists fixes up.

Analysis:



成功大學

COPYRIGHT 2002 NATIONAL CHENG KUNG UNIVERSITY



- ▶ Relies on no bucket getting too many values.
- ▶ All lines of algorithm except insertion sorting take $\Theta(n)$ altogether.
- ▶ Intuitively, if each bucket gets a constant number of elements, it takes $O(1)$ time to sort each bucket $\Rightarrow O(n)$ sort time for all buckets.
- ▶ We “expect” each bucket to have few elements, since the average is 1 element per bucket.
- ▶ But we need to do a careful analysis.

Define a random variable:

► n_i = the number of elements placed in bucket $B[i]$.

Because insertion sort runs in quadratic time, bucket sort time is $T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$

Take expectations of both sides:

$$\begin{aligned} E[T(n)] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} E\left[O(n_i^2)\right] \quad (\text{linearity of expectation}) \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \quad (E[aX] = aE[X]) \end{aligned}$$

Claim

$$E[n_i^2] = 2 - (1/n) \text{ for } i = 0, \dots, n - 1.$$

Proof of claim

Define indicator random variables:

- ▶ $X_{ij} = I\{A[j] \text{ falls in bucket } i\}$
- ▶ $\Pr\{A[j] \text{ falls in bucket } i\} = 1/n$
- ▶ $n_i = \sum_{j=1}^n X_{ij}$



Then

$$\begin{aligned}
 E[n_i^2] &= E\left[\sum_{j=1}^n X_{ij}^2\right] \\
 &= E\left[\sum_{j=1}^n X_{ij}^2 + 2\sum_{j=1}^{n-1} \sum_{k=j+1}^n X_{ij} X_{ik}\right] \\
 &= \sum_{j=1}^n E[X_{ij}^2] + 2\sum_{j=1}^{n-1} \sum_{k=j+1}^n E[X_{ij} X_{ik}] \quad (\text{linearity of expectation})
 \end{aligned}$$



$$\begin{aligned} E[X_{ij}^2] &= 0^2 \cdot \Pr\{A[j] \text{ doesn't fall in bucket } i\} + 1^2 \cdot \Pr\{A[j] \text{ falls in bucket } i\} \\ &= 0 \cdot \left[1 - \frac{1}{n}\right] + 1 \cdot \frac{1}{n} \\ &= \frac{1}{n} \end{aligned}$$

$E[X_{ij}X_{ik}]$ for $j \neq k$: Since $j \neq k$, X_{ij} and X_{ik} are independent random variables

$$\begin{aligned} \Rightarrow E[X_{ij}X_{ik}] &= E[X_{ij}]E[X_{ik}] \\ &= \frac{1}{n} \cdot \frac{1}{n} \\ &= \frac{1}{n^2} \end{aligned}$$



Therefore:

$$\begin{aligned}
 E[n_i^2] &= \sum_{j=1}^n \frac{1}{n} + 2 \sum_{j=1}^{n-1} \sum_{k=j+1}^n \frac{1}{n^2} \\
 &= n \cdot \frac{1}{n} + 2 \left[\begin{matrix} n \\ 2 \end{matrix} \right] \frac{1}{n^2} \\
 &= 1 + \frac{n-1}{n} \\
 &= 1 + 1 - \frac{1}{n} \\
 &= 2 - \frac{1}{n}
 \end{aligned}$$

■

(claim)



Therefore:

$$\begin{aligned} E[T(n)] &= \Theta(n) + \sum_{i=0}^{n-1} O(2^{-1/n}) \\ &= \Theta(n) + O(n) \\ &= \Theta(n) \end{aligned}$$



- ▶ Again, not a comparison sort. Used a function of key values to index into an array.
- ▶ This is a **probabilistic analysis**—we used probability to analyze an algorithm whose running time depends on the distribution of inputs.
- ▶ Different from a **randomized algorithm**, where we use randomization to impose a distribution.
- ▶ With bucket sort, if the input isn't drawn from a uniform distribution on $[0,1)$, all bets are off (performance-wise, but the algorithm is still correct).