

# Algorithms

## Chapter 8

### Sorting in Linear Time

Associate Professor: Ching-Chi Lin

林清池 副教授

[chingchi.lin@gmail.com](mailto:chingchi.lin@gmail.com)

Department of Computer Science and Engineering  
National Taiwan Ocean University

# Outline

---

- ▶ **Lower bounds for sorting**
- ▶ Counting sort
- ▶ Radix sort
- ▶ Bucket sort

# Overview

---

- ▶ Sort  $n$  numbers in  $O(n \lg n)$  time
  - ▶ Merge sort and heapsort achieve this upper bound in the worst case.
  - ▶ Quicksort achieves it on average.
  - ▶ For each of these algorithms, we can produce a sequence of  $n$  input numbers that causes the algorithm to run in  $\Theta(n \lg n)$  time.
- ▶ **Comparison sorting**
  - ▶ The only operation that may be used to gain order information about a sequence is comparison of pairs of elements.
  - ▶ All sorts seen so far are comparison sorts: insertion sort, selection sort, merge sort, quicksort, heapsort.

# Lower bounds for sorting

---

## ▶ **Lower bounds**

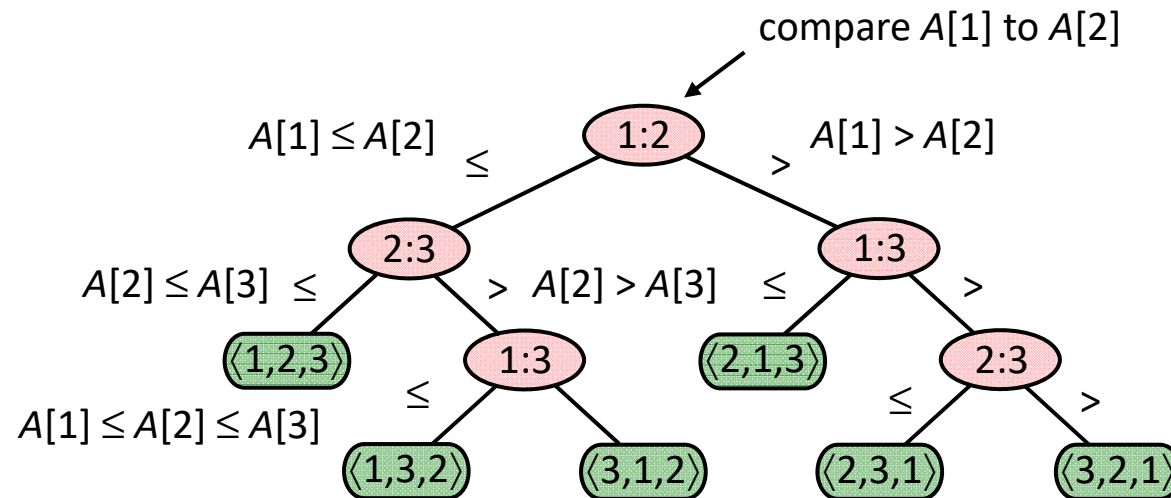
- ▶  $\Omega(n)$  to examine all the input.
- ▶ All sorts seen so far are  $\Omega(n \lg n)$ .
- ▶ We'll show that  $\Omega(n \lg n)$  is a lower bound for comparison sorts.

## ▶ **Decision tree**

- ▶ Abstraction of any comparison sort.
- ▶ A full binary tree.
- ▶ Represents comparisons made by
  - ▶ a specific sorting algorithm
  - ▶ on inputs of a given size.
- ▶ Control, data movement, and all other aspects of the algorithm are ignored.

# Decision tree

- ▶ For insertion sort on 3 elements:



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

# Properties of decision trees<sub>1/3</sub>

- ▶ **Lemma 1** Any binary tree of height  $h$  has  $\leq 2^h$  leaves.
  - ▶ *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.
    - ▶ # of leaves for height  $h = 2 \cdot (\text{\# of leaves for height } h - 1)$ 

$= 2 \cdot 2^{h-1}$  (induction hypothesis)

$= 2^h$ .

## Properties of decision trees<sub>2/3</sub>

---

- ▶ **Theorem 1** Any decision tree that sorts  $n$  elements has height  $\Omega(n \lg n)$ .

*Proof:*

- ▶  $\ell \geq n!$ , where  $\ell$  = # of leaves.
- ▶ By lemma 1,  $n! \leq \ell \leq 2^h$  or  $2^h \geq n!$ .
- ▶ Take logs:  $h \geq \lg(n!)$ .
- ▶ Use Stirling's approximation:  $n! > (n/e)^n$

$$\begin{aligned} h &\geq \lg(n!) \\ &> \lg(n/e)^n \\ &= n \lg(n/e) \\ &= n \lg n - n \lg e \\ &= \Omega(n \lg n). \end{aligned}$$

## Properties of decision trees<sub>3/3</sub>

---

- ▶ **Corollary 1** Heapsort and merge sort are asymptotically optimal comparison sorts.

*Proof:*

- ▶ The  $O(n \lg n)$  upper bounds on the running times for heapsort and merge sort match the  $\Omega(n \lg n)$  worst-case lower bound from Theorem 1.



# Outline

---

- ▶ Lower bounds for sorting
- ▶ **Counting sort**
- ▶ Radix sort
- ▶ Bucket sort

# Counting sort

---

- ▶ Non-comparison sorts.
- ▶ Depends on a **key assumption**: numbers to be sorted are integers in  $\{0, 1, \dots, k\}$ .
- ▶ **Input**:  $A[1 \dots n]$ , 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**:  $B[1 \dots n]$ , sorted.  $B$  is assumed to be already allocated and is given as a parameter.
- ▶ **Auxiliary storage**:  $C[0 \dots k]$ .
- ▶ **Worst-case running time**:  $\Theta(n+k)$ .

# The COUNTING-SORT procedure

---

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

```
1.  for  $i \leftarrow 0$  to  $k$ 
2.      do  $C[i] \leftarrow 0$  }  $\Theta(k)$ 
3.  for  $j \leftarrow 1$  to  $length[A]$ 
4.      do  $C[A[j]] \leftarrow C[A[j]] + 1$  }  $\Theta(n)$ 
5.  /*  $C[i]$  now contains the number of elements equal to  $i$ . */
6.  for  $i \leftarrow 1$  to  $k$ 
7.      do  $C[i] \leftarrow C[i] + C[i - 1]$  }  $\Theta(k)$ 
8.  /*  $C[i]$  now contains the number of elements less than or equal to  $i$ . */
9.  for  $j \leftarrow length[A]$  downto 1
10.     do  $B[C[A[j]]] \leftarrow A[j]$ 
11.     do  $C[A[j]] \leftarrow C[A[j]] - 1$  }  $\Theta(n)$ 
```

► **The running time:**  $\Theta(n+k)$ .

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



	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	0	2	3	0	1

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

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



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



	1	2	3	4	5	6	7	8
B		0					3	

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

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

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

## Properties of counting sort

---

- ▶ A sorting algorithm is said to be **stable** if keys with same value appear in same order in output as they did in input.
- ▶ **Counting sort is stable** because of how the last loop works.
- ▶ Counting sort will be used in radix sort.

# Outline

---

- ▶ Lower bounds for sorting
- ▶ Counting sort
- ▶ **Radix sort**
- ▶ Bucket sort

# Radix sort

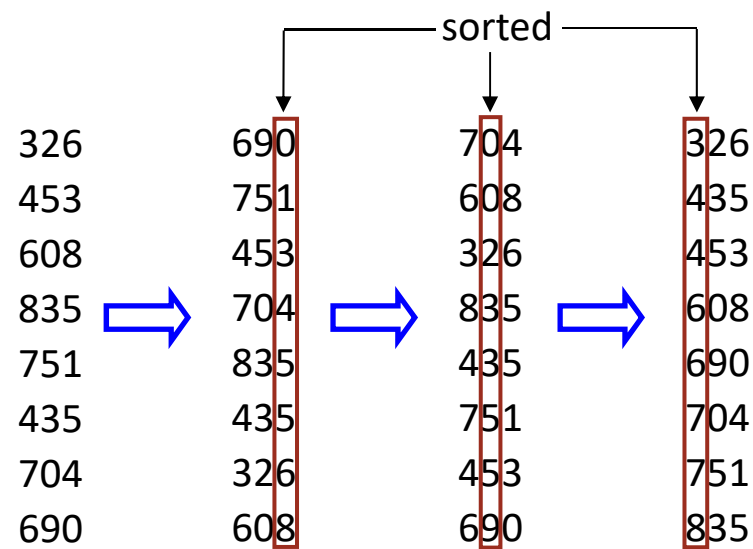
---

- ▶ **Key idea:** Sort **least** significant digits first.

RADIX-SORT( $A, d$ )

1. **for**  $i \leftarrow 1$  **to**  $d$
2.     **do** use a stable sort to sort array  $A$  on digit  $i$

- ▶ An example:



# Correctness of radix sort

---

- ▶ **Proof:** By induction on number of passes ( $i$  in pseudocode).
- ▶ **Basis:**
  - ▶  $i = 1$ . There is only one digit, so sorting on that digit sorts the array.
- ▶ **Inductive step:**
  - ▶ Assume digits 1, 2, ...,  $i - 1$  are sorted.
  - ▶ Show that a stable sort on digit  $i$  leaves digits 1, 2, ...,  $i$  sorted:
    - ▶ If 2 digits in position  $i$  are different, ordering by position  $i$  is correct, and positions 1, ...,  $i - 1$  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.



## Time complexity of radix sort

---

- ▶ Assume that we use counting sort as the intermediate sort.
- ▶ When each digit is in the range 0 to  $k-1$ , each pass over  $n$   $d$ -digit number takes time  $\Theta(n + k)$ .
- ▶ There are  $d$  passes, so the total time for radix sort is  $\Theta(d(n + k))$ .
- ▶ If  $k = O(n)$ , time =  $\Theta(dn)$ .
- ▶ **Lemma 2:** Given  $n$   $d$ -digit numbers in which each digit can take on up to  $k$  possible values, RADIXSORT correctly sorts these numbers in  $\Theta(d(n + k))$  time.

## Break each key into digits

---

- ▶ **Lemma 3:** Given  $n$   $b$ -bit numbers and any positive integer  $r \leq b$ , RADIX-SORT correctly sorts these numbers in  $\Theta((b/r)(n + 2^r))$  time.
- ▶ **Proof**
  - ▶ We view each key as having  $d = \lceil b/r \rceil$  digits of  $r$  bits each.
  - ▶ Each digit is an integer in the range 0 to  $2^r - 1$ , so that we can use counting sort with  $k = 2^r$ .
  - ▶ Each pass of counting sort takes time  $\Theta(n+k) = \Theta(n+2^r)$ .
  - ▶ A total running time of  $\Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$ .
- ▶ **For example:**
  - ▶ 32-bit words, 8-bit digits.
  - ▶  $b = 32, r = 8, d = 32/8 = 4, k = 2^8 - 1 = 255$ .

# The main reason

---

- ▶ 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.

# Outline

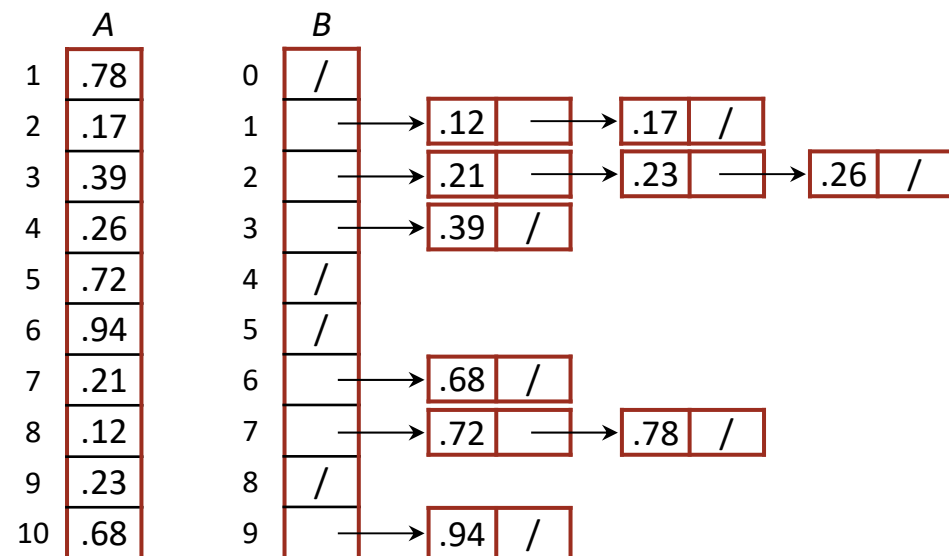
---

- ▶ Lower bounds for sorting
- ▶ Counting sort
- ▶ Radix sort
- ▶ **Bucket sort**

# Bucket sort

---

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



# The BUCKET SORT procedure

---

- ▶ **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( $A, n$ )

1. **for**  $i \leftarrow 1$  **to**  $n$
2.     **do** insert  $A[i]$  into list  $B[\lfloor n \cdot A[i] \rfloor]$
3. **for**  $i \leftarrow 0$  **to**  $n - 1$
4.     **do** sort list  $B[i]$  with insertion sort
5. concatenate lists  $B[0], B[1], \dots, B[n - 1]$  together in order
6. **return** the concatenated lists

## Correctness of bucket sort

---

- ▶ 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.

## Time complexity of bucket sort

---

- ▶ 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.



# Time complexity of bucket sort

- ▶ 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:

$$E[T(n)] = E\left[\theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \quad \text{Claim that } E[n_i^2] = 2 - 1/n \text{ for } 0 \leq i \leq n-1.$$

$$\begin{aligned} &= \theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \\ &= \theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \end{aligned} \quad \begin{array}{l} \text{Therefore, } E[T(n)] = \theta(n) + \sum_{i=0}^{n-1} O(2 - 1/n) \\ = \theta(n) + O(n) \\ = \theta(n). \end{array}$$

linearity of expectation

$E[aX] = aE[X]$

## Proof of claim

---

▶ **Claim:**  $E[n_i^2] = 2 - 1/n$  for  $0 \leq i \leq n - 1$ .

▶ **Proof**

- ▶  $\Pr\{A[j] \text{ falls in bucket } i\} = p = 1/n$ .
- ▶ The probability that  $n_i = k$  follows the binomial distribution  $b(k; n, p)$ .
- ▶ So,  $E[n_i] = np = 1$  and variance  $\text{Var}[n_i] = np(1 - p) = 1 - 1/n$ .
- ▶ For any random variable  $X$ , we have  $E[n_i^2] = \text{Var}[n_i] + E^2[n_i]$

$$\begin{aligned} &= 1 - \frac{1}{n} + 1^2 \\ &= 2 - \frac{1}{n}. \end{aligned}$$

# Notes

---

- ▶ 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).