Linear Sorting

Chapter 8

So far…

- Introduced sorting algorithms that sort in O(nlgn)
  - Merge sort: worst case
  - Heapsort: worst case
  - Quicksort: average case

- Lower bound on these algorithms is Ω(nlgn)

Decision Tree

- We can show this lower bound using decision trees

Comparison Sorts

- All these algorithms share an interesting property:
  - The sorted order they determine is based only on comparison between the elements

- Comparison Sorts!

Counting Sort

- Depends on a key assumption:
  - numbers to be sorted are integers in $\{0, 1, \ldots, k\}$
- **Input:** A[1..n]
- **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, k)

```
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

Example

- $2_1, 5_1, 3_1, 0_1, 2_2, 3_2, 0_2, 3_3$

Analysis

- Is counting sort stable?
  - What does stable mean?
- Analysis:

- How big of k is practical?

Your Turn

- A: <6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2>

Radix Sort

- How IBM made its money. Punch card readers for census tabulation in early 1900's. Card sorters, worked on one column at a time. It's the algorithm for using the machine that extends the technique to multi-column sorting. The human operator was part of the algorithm!

- We're going to sort d digits

RADIX-SORT(A, d)

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

	one's place	ten's place	100s place
329			
457			
657			
839			
436			

Bucket Sort

- Assumption: input is generated by a random process that distributes elements uniformly over [0,1)

- Idea:

Bucket Sort

- Input: A[1..n], where for all i

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

BUCKET-SORT(A)

BUCKET-SORT(A)

```
n = A.length
let B[0..n − 1] be a new array
for i = 0 to n − 1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n − 1
    sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

Example

- A:<.78, .17, .39, .26, .72, .94, .21, .12, .23, .68>