Linear Sorts

Linear Sorts?

Comparison sorts are very general, but are $\Omega(n \log n)$

Faster sorting may be possible if we can constrain the nature of the input.

Linear Sorting Algorithms

- Counting Sort
- Radix Sort
- Bucket Sort

Linear Sorts: Learning Outcomes

- From understanding this lecture you should be able to:
  - Explain the difference between comparison sorts and linear sorting methods.
  - Identify situations when linear sorting methods can be applied and know why.
  - Explain and code any of the linear sorting algorithms we have covered.

Linear Sorting Algorithms

- **Counting Sort**
- Radix Sort
- Bucket Sort

Example 1. Counting Sort

- Invented by Harold Seward in 1954.
- Counting Sort applies when the elements to be sorted come from a finite (and preferably small) set.
- For example, the elements to be sorted are integers in the range [0…k-1], for some fixed integer k.
- We can then create an array V[0…k-1] and use it to count the number of elements with each value [0…k-1].
- Then each input element can be placed in exactly the right place in the output array in constant time.

Counting Sort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	3	3	3

- Input: N records with integer keys between [0…3].
- Output: Stable sorted keys.
- Algorithm:
  - Count frequency of each key value to determine transition locations
  - Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Stable sort: If two keys are the same, their order does not change.

Thus the 4th record in input with digit 1 must be
the 4th record in output with digit 1.

It belongs at output index 8, because 8 records go before it

ie, 5 records with a smaller digit & 3 records with the same digit

Count These!

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:																			
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
# of records with digit v:	5	9	3	2

N records. Time to count? $\theta(N)$

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:																			
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
# of records with digit v:	5	9	3	3
# of records with digit < v:	0	5	14	17

N records, k different values. Time to count? $\theta(k)$

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
# of records with digit < v:	0	5	14	17

= location of first record with digit v.

End of Lecture

Nov 26, 2015

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	?				1													
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of first record with digit v.	0	5	14	17

Algorithm: Go through the records in order putting them where they go.

Loop Invariant

- The first *i-1* keys have been placed in the correct locations in the output array
- The auxiliary data structure *v* indicates the location at which to place the i^{th} key for each possible key value from *[0..k-1]*.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:						1													
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of **next** record with digit v.	0	5	14	17

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0					1													
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	0	6	14	17

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1													
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	1	6	14	17

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1												
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	2	6	14	17

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	~~1~~	~~0~~	~~0~~	~~1~~	~~3~~	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1											3	
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	2	7	14	17

Algorithm: Go through the records in order putting them where they go.

CountingSort

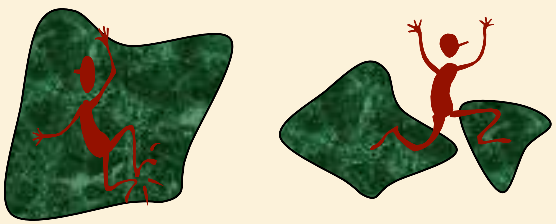
Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1	1										3	
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of **next** record with digit v.	2	7	14	18

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1	1	1									3	
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18



Value v:	0	1	2	3
	2	8	14	18

Location of next record with digit v.

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	~~1~~	~~0~~	~~0~~	~~1~~	~~3~~	~~1~~	~~1~~	~~3~~	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1	1	1									3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	2	9	14	18

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	~~1~~	~~0~~	~~0~~	~~1~~	~~3~~	~~1~~	~~1~~	~~3~~	~~1~~	0	2	1	0	1	1	2	2	1	0
Output:	0	0				1	1	1	1	1								3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	2	9	14	19

Location of next record with digit v.

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0			1	1	1	1	1								3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	2	10	14	19

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0			1	1	1	1	1					2			3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	3	10	14	19

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0			1	1	1	1	1	1				2			3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	3	10	15	19

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0			1	1	1	1	1	1				2			3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of **next** record with digit v.	3	10	15	19

Algorithm: Go through the records in order putting them where they go.

CountingSort

Input:	1	0	0	1	3	1	1	3	1	0	2	1	0	1	1	2	2	1	0
Output:	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	3	3
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Value v:	0	1	2	3
Location of next record with digit v.	5	14	17	19

Time = $\theta(N)$

Total = $\theta(N+k)$

Linear Sorting Algorithms

- Counting Sort
- **Radix Sort**
- Bucket Sort

Example 2. RadixSort

Input:

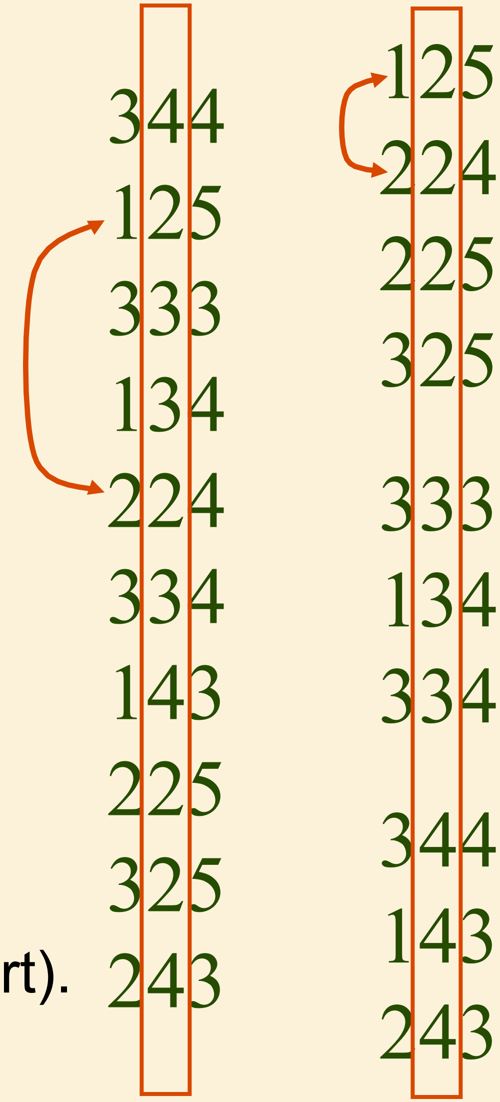
- An array of *N* numbers.
- Each number contains *d* digits.
- Each digit between *[0…k-1]*

Output:

- Sorted numbers.

Digit Sort:

- Select one digit
- Separate numbers into k piles based on selected digit (e.g., Counting Sort).



Stable sort: If two cards are the same for that digit, their order does not change.

RadixSort

344	125	125
125	134	224
333	143	225
134	224	325
224	225	134
334	243	333
143	344	334
225	333	143
325	334	243
243	325	344

Sort by which digit first?

The most significant.

Sort by which digit Second?

The next most significant.

All meaning in first sort lost.

RadixSort

344
125
333
134
224
334
143
225
325
243

Sort by which digit first?

The least significant.

333
143
243
344
134
224
334
125
225
325

Sort by which digit Second?

The next least significant.

224
125
225
325
333
134
334
143
243
344

RadixSort

344
125
333
134
224
334
143
225
325
243

Sort by which digit first?

The least significant.

333
143
243
344
134
224
334
125
225
325

Sort by which digit Second?

The next least significant.

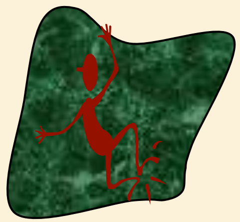
2 24
1 25
2 25
3 25
3 33
1 34
3 34
1 43
2 43
3 44

Is sorted by least sig. 2 digits.

RadixSort

2 24
1 25
2 25
3 25
3 33
1 34
3 34
1 43
2 43
3 44

i+1



Is sorted by first i digits.

Sort by i+1st digit.

1 25
1 34
1 43

2 24
2 25
2 43

3 25
3 33
3 34
3 44

Is sorted by first i+1 digits.

These are in the correct order because sorted by high order digit

RadixSort

2 24
1 25
2 25
3 25
3 33
1 34
3 34
1 43
2 43
3 44

i+1

Is sorted by first i digits.

Sort by i+1st digit.

1 | 25
1 | 34
1 | 43

2 | 24
2 | 25
2 | 43

3 | 25
3 | 33
3 | 34
3 | 44

Is sorted by first i+1 digits.

These are in the correct order because was sorted & stable sort left sorted

Loop Invariant

- The keys have been correctly stable-sorted with respect to the *i-1* least-significant digits.

Running Time

RADIX-SORT(A, d)

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

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

Running time is $\Theta(d(n+k))$

Where

$d =$ # of digits in each number

$n =$ # of elements to be sorted

$k =$ # of possible values for each digit

Linear Sorting Algorithms

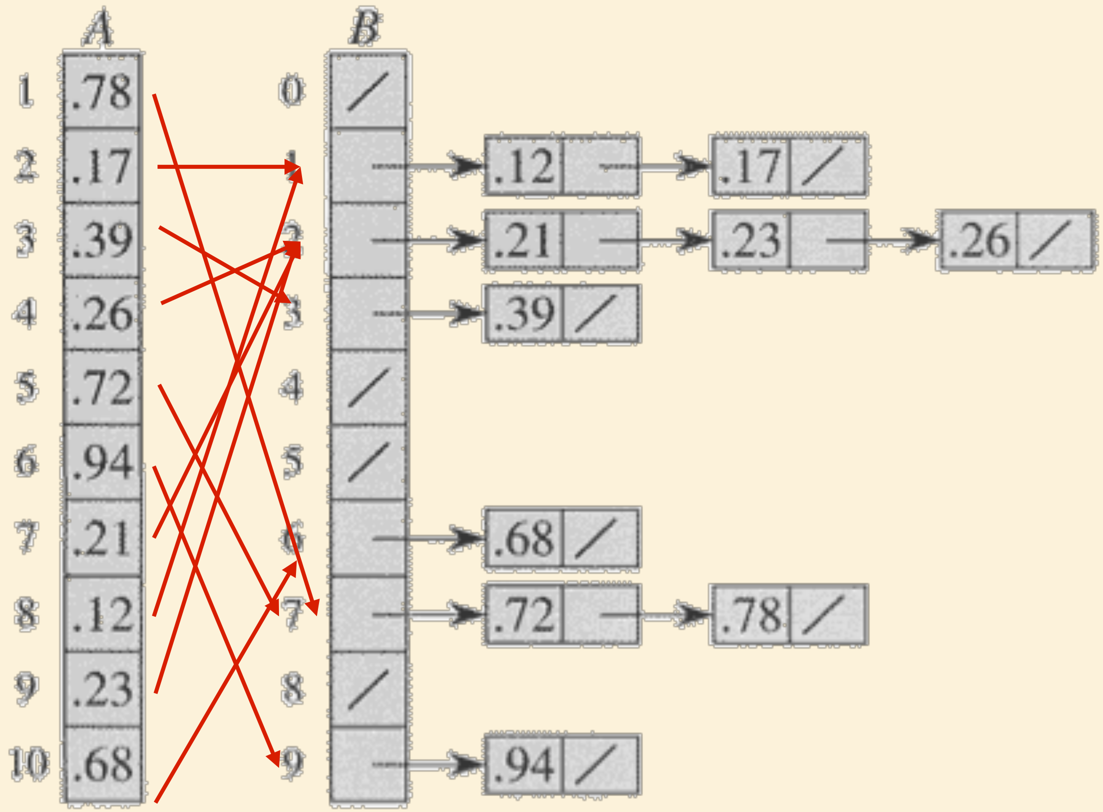
- Counting Sort
- Radix Sort
- **Bucket Sort**

Example 3. Bucket Sort

- Applicable if input is constrained to finite interval, e.g., real numbers in the range [0…1).
- If input is random and uniformly distributed, **expected** run time is $\Theta(n)$.

Bucket Sort

- Given A[1..n]:
  - Create new table *B* of length *n*
  - Insert A[i] into $B\left[\left\lfloor nA[i]\right\rfloor\right]$



PseudoCode

BUCKET-SORT(A, n)

Expected Running Time

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

  do insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$ ⟵ $\Theta(1) \times n$

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

  do sort list $B[i]$ with insertion sort ⟵ $\Theta(1) \times n$

concatenate lists $B[0], B[1], \ldots, B[n-1]$ ⟵ $\Theta(n)$

return the concatenated lists

$\Theta(n)$

Loop Invariants

- Loop 1
  - The first *i-1* keys have been correctly placed into buckets of width *1/n*.
- Loop 2
  - The keys within each of the first *i-1* buckets have been correctly stable-sorted.

Linear Sorting Algorithms

- Counting Sort
- Radix Sort
- Bucket Sort

Linear Sorts: Learning Outcomes

- From understanding this lecture you should be able to:
  - Explain the difference between comparison sorts and linear sorting methods.
  - Identify situations when linear sorting methods can be applied and know why.
  - Explain and code any of the linear sorting algorithms we have covered.