Linear Sorts

Chapter 11
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 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 $[0 \ldots k-1]$, for some fixed integer $k$.

- We can then create an array $V[0 \ldots k-1]$ and use it to count the number of elements with each value $[0 \ldots k-1]$.

- Then each input element can be placed in exactly the right place in the output array in constant time.
Counting Sort

Input: $N$ records with integer keys between $[0 \ldots 3]$.

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.

<table>
<thead>
<tr>
<th>Input:</th>
<th>1 0 0 1 3 1 1 3 1 0 2 1 0 1 1 2 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 3 3 3</td>
</tr>
</tbody>
</table>
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 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 |

<table>
<thead>
<tr>
<th>Value v:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td># of records with digit v:</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

N records. Time to count? $\Theta(N)$
## CountingSort

<table>
<thead>
<tr>
<th>Input:</th>
<th>1 0 0 1 3 1 1 3 1 0 2 1 0 1 1 2 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td></td>
</tr>
<tr>
<td>Index:</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Input:</th>
<th>1 0 0 1 3 1 1 3 1 0 2 1 0 1 1 2 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 3 3</td>
</tr>
<tr>
<td>Index:</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18</td>
</tr>
</tbody>
</table>

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

= location of first record with digit v.
## 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 |
         | 0 5 14 17 |

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 | 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 |

| 0 | 6 | 14 | 17 |

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

Value v:  

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>14</td>
<td>17</td>
</tr>
</tbody>
</table>

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
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.

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

<table>
<thead>
<tr>
<th>Input:</th>
<th>1 0 0 1 3 1 1 3 1 0 2 1 0 1 1 2 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>Index:</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

Value v:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>14</td>
<td>17</td>
</tr>
</tbody>
</table>

Location of next record with digit v.

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

**Input:**

<p>| | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Output:**

<p>| | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.**

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
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.

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 |

| 2 | 9 | 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 |
| 2 | 9 | 14 | 19 |

**Location of next record with digit v.**

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>14</td>
<td>19</td>
</tr>
</tbody>
</table>

**Location of next record with digit v.**

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

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

Location of next record with digit v.

Value v: 0 1 2 3
     3 10 14 19
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:

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 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:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>

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 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:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>14</td>
<td>17</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

Location of next record with digit v.

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

Sort wrt which digit first?
The most significant.

Sort wrt which digit Second?
The next most significant.

All meaning in first sort lost.
### RadixSort

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>344</td>
<td>125</td>
<td>333</td>
</tr>
<tr>
<td>333</td>
<td>134</td>
<td>224</td>
</tr>
<tr>
<td>224</td>
<td>334</td>
<td>143</td>
</tr>
<tr>
<td>334</td>
<td>143</td>
<td>243</td>
</tr>
<tr>
<td>143</td>
<td>225</td>
<td>325</td>
</tr>
<tr>
<td>225</td>
<td>243</td>
<td>344</td>
</tr>
<tr>
<td>325</td>
<td>344</td>
<td></td>
</tr>
</tbody>
</table>

**Sort wrt which digit first?**

- The least significant.

**Sort wrt which digit Second?**

- The next least significant.
### RadixSort

<table>
<thead>
<tr>
<th>Original</th>
<th>Sorted wrt least sig. 2 digits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>344</td>
<td>333</td>
</tr>
<tr>
<td>125</td>
<td>143</td>
</tr>
<tr>
<td>333</td>
<td>243</td>
</tr>
<tr>
<td>134</td>
<td>344</td>
</tr>
<tr>
<td>224</td>
<td>134</td>
</tr>
<tr>
<td>334</td>
<td>224</td>
</tr>
<tr>
<td>143</td>
<td>334</td>
</tr>
<tr>
<td>225</td>
<td>125</td>
</tr>
<tr>
<td>325</td>
<td>225</td>
</tr>
<tr>
<td>243</td>
<td>325</td>
</tr>
</tbody>
</table>

Sort wrt which digit first?

Sort wrt which digit Second?

The least significant.

The next least significant.

Is sorted wrt least sig. 2 digits.
RadixSort

Sort wrt i+1st digit.

Is sorted wrt first i digits.

These are in the correct order because sorted wrt high order digit.
RadixSort

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

Is sorted wrt first i digits.

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

Is sorted wrt first i+1 digits.

Sort wrt i+1st digit.

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

**Algorithm**

```
RADIX-SORT(A, d)
for i ← 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

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

**Bucket-Sort**\(_{(A,n)}\)

for \(i \leftarrow 1\) to \(n\)

\(\text{do}\) insert \(A[i]\) into list \(B[\lfloor n \cdot A[i] \rfloor]\)

for \(i \leftarrow 0\) to \(n - 1\)

\(\text{do}\) sort list \(B[i]\) with insertion sort

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

return the concatenated lists

---

**Expected Running Time**

\(\Theta(1) \times n\)

\(\Theta(1) \times n\)

\(\Theta(n)\)

\(\Theta(n)\)
Linear Sorting Algorithms

- Counting Sort
- Radix Sort
- Bucket Sort
Linear Sorts: Learning Outcomes

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/or code any of the linear sorting algorithms we have covered.