Counting sort: ideas

- **Problem:** to sort $A$, an array of integers
- **Extra info:** each of the $n$ input elements of $E$ is an integer in the range $[0, k]$.
- **One scan through $A$** will find how many times each integer $i \in [0, k]$ appears in $A$. The counts are stored in an auxiliary array $C[0..k]$.
- **Thus element $C[i]$** is the number of times that integer $i$ appears in $A$. $\sum_{j=0}^{i} C[i]$ gives the number of elements of $A$ that are less than $i$, and this tells where to put $i$ in a sorted array.
Counting sort: algorithm

Counting-sort (A, B, k)
1. for (i=0; i<k; i++)
2.     C[i] = 0; // initialize array C to store counts.
3. for (j=0; j<n; j++)
5. for (l = 1, i<k, i++)
6.     C[i] = C[i] + C[i-1]; // transform to cumulative counts
7. for (j = n; j > 1; j--)
8.     B[C[A[j]]] = A[j]; // sort to the right place

Counting sort: example

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
<th>5</th>
<th>3</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

(a) (b) (c) (d) (e)
Counting sort: complexity analysis

- Time

\[ \Theta(k) + \Theta(n) + \Theta(k) + \Theta(n) \subseteq \Theta(k + n) \]

- Space: \(\Theta(k+n)\) not in-place

Note:
1. Counting sort is not comparison based sorting. Instead, it uses the actual values of the elements to index into an array. Thus, its linear performance breaks the \(\Omega(n \log n)\) lower bound for comparison based sorting problems.
2. Stable sort

Radix sort: ideas

- Problem: to sort \(A\), an array of \(n\) \(d\)-digit numbers

- Strategy: sort numbers based on their least significant digit, then on their second least significant digit, so on and so forth, until sort on their most significant digit is done.
### Radix sort: an example

<table>
<thead>
<tr>
<th>unsorted</th>
<th>ones</th>
<th>tens</th>
<th>hundreds</th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>720</td>
<td>720</td>
<td>329</td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>329</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>436</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>457</td>
<td>839</td>
<td>457</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>355</td>
<td>657</td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>457</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>657</td>
<td>839</td>
</tr>
</tbody>
</table>

Correctness: Induction on digit position

Assume numbers are sorted up to the i-th digit. When sort on (i+1) -th digit:

1) Two numbers that differ in the (i+1)-th digit are correctly sorted.
2) Two numbers having same (i+1)-th digit remain the same order up to i-th digit, because sorting on each digit is stable.

---

### Radix sort:

- Why not sort from most significant digits to less significant digits?

```
1 3   4 2
4 2   1 3
```

Radix sort: analysis

Time = (number of digits) x (time for sorting on each digit)
= d x O(10+n) \subseteq O(n)

note: counting sort is utilized for sorting on each digit.

Exercise: Show how to sort n integers in the range 1 to n² in O(n) time.

More sorting?

Bounds for sorting by prefix reversal.
Discrete Mathematics 27, 47–57, 1979

The authors study the problem of sorting a sequence of distinct numbers by prefix reversal -- reversing a subsequence of adjacent numbers which always contains the first number of the current sequence. Let f(n) denote the smallest number of prefix reversals which is sufficient to sort n numbers in any ordering. The authors prove that f(n) \leq (5n+5)/3 by demonstrating an algorithm which never needs more prefix reversals. They also prove that f(n) \geq 17n/16 whenever n is a multiple of 16. The sequences which achieve this bound are periodic extensions of the basic sequence 1, 7, 5, 3, 6, 4, 2, 8, 16, 10, 12, 14, 11, 13, 15, 9. If, furthermore, each integer is required to participate in an even number of prefix reversals, the corresponding function g(n) is shown to satisfy 3n/2 - 1 \leq g(n) \leq 2n+3.