Counting sort treats the element as just one digit

Radix Sort

From least to most significant digit $d = 0$ to $b - 1$

apply counting sort where we access digit $d$
ex

329
457
657
839
436
720
355

720
436
457
355
329
436
457
657
839

phase 0: using counting sort on digit 0

phase 1: look at place 1 of 839

place 1: 839

place 2: 720

place 3: 329

place 4: 436

place 5: 457

place 6: 657

place 7: 839

place 8: 329

place 9: 436

place 10: 457

place 11: 657

place 12: 839
Radix

CountingSort(input, output, k) {
    for (d = 0 to #digits - 1)
    {
        int n = input.length;
        For (i = 0; i < k; i++)
            count[i] = 0;
        For (j = 0; j < n; j++)
            count[input[j]]++;
        For (i = 1; i < k; i++)
            count[i] += count[i - 1];
        For (j = n - 1; j > 0; j --)
            output[--count[input[j]]] = input[j];
    }
    digitizer.getDigit(input[j], d) ➔ current digit
Correctness Highlights.

Prove following holds using induction.

After first $p$ phases, numbers when looking at least significant $p$ digits are sorted.

Base: true after 1 phase by counting sort.

Inductive step: combines inductive hyp.

Correctness of counting sort + stability of counting sort.
What's the asymptotic time complexity?

\( \Theta(d(n+b)) \)
Return to the ex.

\[ N = \# \text{ social security } \#s \text{ to sort} \]
\[ d = 9 \quad \text{(treat each base-10 digit as a digit for radix sort)} \]
\[ b = 10 \]

\[ \text{(9}(n+10) = \text{(H}(n) \]
\[ \Theta(9(n+10)) \]

\[ \underbrace{XXX \quad XXX \quad XXX} \]

base 1000 digit digit number

\[ 0, \ldots, 999 \]

\[ \text{roughly 3 times faster for "large" } n \]

Time complexity with this digitizer

\[ \Theta(3(n+1000)) \]
Consider sorting #s that begin in binary (base 2)

n #s
b bits (b=32, 32-bit #)

group [b] bits into a digit

# digits = \( \frac{b}{r} \)  \quad \text{base per digit} = 2^r
$\Theta \left( \frac{b}{r} (n + 2^r) \right)$

$\frac{C \cdot D}{r} (n + 2^r)$

Could even work out constants more carefully.

Min with respect to $r$.

Roughly optimizes when $r = \Theta(\log_2 n)$
ADT Taxonomy Part II

Plan For today
- Finish discussion of radix sort
- Briefly discuss bucket sort
- Return to ADT Taxonomy
- Introduce Set ADT (if time permits)
protected void radixsortImpl(Digitizer<? super E> digitizer) {
    Object[] from = new Object[getSize()];
    Object[] to = new Object[getSize()];
    int b = digitizer.getBase();
    int count[] = new int[b];
    int numDigits = 0;
    for (int i = 0; i < getSize(); i++) {
        from[i] = a[getPosition(i)];
        numDigits = max(numDigits, digitizer.numDigits((E) from[i]));
    }
    for (int d = 0; d < numDigits; d++) {
        Arrays.fill(count, 0);
        for (Object x : from)
            count[digitizer.getDigit((E) x, d)]++;
        for (int i = 1; i < b; i++)
            count[i] += count[i-1];
        for (int i = getSize() - 1; i >= 0; i--)
            to[-count[digitizer.getDigit((E) from[i], d)]] = from[i];
        Object[] temp = from; from = to; to = temp;  // swap the "from" and "to" arrays
    }
    for (int i = 0; i < getSize(); i++)
        put(getPosition(i), from[i]);
    version.increment();  // invalidate locators for iteration
}
Optimizing radix sort

time complexity \( C \cdot \frac{b}{r} \left( n + 2^r \right) \)

\# bits/dig. \quad \# \text{bits}

Intuitively want to set \( r \) so \( n = 2^r \) so you reduce \# digits while not making base too high

Solving for \( r \) in \( n = 2^r \) yields \( r = \log_2 n \)
Example: let \( c = 5 \), \( n = 1,000,000 \)

<table>
<thead>
<tr>
<th># bits/digit</th>
<th># digits</th>
<th>base^{e_k}</th>
<th>c \cdot d(n+k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic radix sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>2</td>
<td>160,000,300</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>4</td>
<td>80,000,320</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>40,000,640</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>256</td>
<td>20,005,120</td>
</tr>
<tr>
<td>counting sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>65536</td>
<td>10,655,360</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>( &gt;4 \times 10^9 )</td>
<td>( &gt;4,000,000,000 )</td>
</tr>
</tbody>
</table>
What are the limitations of radix sort?

Requires that you can digitize all elements.

Instead of using a comparator to compare entire elements, a digitizer is used to extract each digit. Think about these costs.