05-0: **Counting Sort**

- Sorting a list of \( n \) integers
- We know all integers are in the range \( 0 \ldots m \)
- We can potentially sort the integers faster than \( n \cdot \log n \)
- Keep track of a “Counter Array” \( C \):
  - \( C[i] = \# \) of times value \( i \) appears in the list

Example: \( 3\ 1\ 3\ 5\ 2\ 1\ 6\ 7\ 8\ 1 \)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

05-1: **Counting Sort Example**

\( 3\ 1\ 3\ 5\ 2\ 1\ 6\ 7\ 8\ 1 \)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

05-2: **Counting Sort Example**

\( 1\ 3\ 5\ 2\ 1\ 6\ 7\ 8\ 1 \)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

05-3: **Counting Sort Example**

\( 3\ 5\ 2\ 1\ 6\ 7\ 8\ 1 \)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

05-4: **Counting Sort Example**

\( 5\ 2\ 1\ 6\ 7\ 8\ 1 \)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

05-5: **Counting Sort Example**
216781

0 1 0 2 0 1 0 0 0

0 1 2 3 4 5 6 7 8 9

05-6: Counting Sort Example

16781

0 1 1 2 0 1 0 0 0

0 1 2 3 4 5 6 7 8 9

05-7: Counting Sort Example

6781

0 2 1 2 0 1 0 0 0

0 1 2 3 4 5 6 7 8 9

05-8: Counting Sort Example

781

0 2 1 2 0 1 1 0 0

0 1 2 3 4 5 6 7 8 9

05-9: Counting Sort Example

81

0 2 1 2 0 1 1 1 0

0 1 2 3 4 5 6 7 8 9

05-10: Counting Sort Example

1

0 2 1 2 0 1 1 1 1 0

0 1 2 3 4 5 6 7 8 9

05-11: Counting Sort Example
### 05-12: Counting Sort Example

<table>
<thead>
<tr>
<th>0</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
<td>9</td>
</tr>
</tbody>
</table>

### 05-13: $\Theta()$ of Counting Sort

- What is the running time of Counting Sort?
- If the list has $n$ elements, all of which are in the range $0 \ldots m$:

  - Running time is $\Theta(n + m)$

- What about the $\Omega(n \lg n)$ bound for all sorting algorithms?

### 05-14: $\Theta()$ of Counting Sort

- What is the running time of Counting Sort?
- If the list has $n$ elements, all of which are in the range $0 \ldots m$:
  - Running time is $\Theta(n + m)$

- What about the $\Omega(n \lg n)$ bound for all sorting algorithms?
  - For *Comparison Sorts*, which allow for sorting arbitrary data. What happens when $m$ is very large?

### 05-15: $\Theta()$ of Counting Sort

- What is the running time of Counting Sort?
- If the list has $n$ elements, all of which are in the range $0 \ldots m$:
  - Running time is $\Theta(n + m)$

- What about the $\Omega(n \lg n)$ bound for all sorting algorithms?
  - For *Comparison Sorts*, which allow for sorting arbitrary data. What happens when $m$ is very large?

### 05-16: Binsort

- Counting Sort will need some modification to allow us to sort *records* with integer keys, instead of just integers.
- Binsort is much like Counting Sort, except that in each index $i$ of the counting array $C$:
  - Instead of storing the *number* of elements with the value $i$, we store a *list* of all elements with the value $i$. 
05-17: Binsort Example

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>9</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>mark</td>
<td>john</td>
<td>mary</td>
<td>sue</td>
<td>julie</td>
<td>rachel</td>
<td>pixel</td>
<td>shadow</td>
<td>alex</td>
<td>james</td>
</tr>
</tbody>
</table>

05-18: Binsort Example

<table>
<thead>
<tr>
<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>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>

05-19: Binsort Example

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>9</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>mark</td>
<td>john</td>
<td>mary</td>
<td>sue</td>
<td>julie</td>
<td>rachel</td>
<td>pixel</td>
<td>shadow</td>
<td>alex</td>
<td>james</td>
</tr>
</tbody>
</table>

05-20: Bucket Sort

- Expand the “bins” in Bin Sort to “buckets”
- Each bucket holds a range of key values, instead of a single key value
- Elements in each bucket are sorted.

05-21: **Bucket Sort Example**

<table>
<thead>
<tr>
<th>114</th>
<th>John</th>
<th>26</th>
<th>Mary</th>
<th>50</th>
<th>Julie</th>
<th>180</th>
<th>Mark</th>
<th>44</th>
<th>Shadow</th>
<th>111</th>
<th>Rachel</th>
<th>4</th>
<th>Pixel</th>
<th>95</th>
<th>Sue</th>
<th>196</th>
<th>James</th>
<th>170</th>
<th>Alex</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<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>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- 0-19
- 20-39
- 40-59
- 60-79
- 80-99
- 100-119
- 120-139
- 140-159
- 160-179
- 180-199

05-22: **Bucket Sort Example**

<table>
<thead>
<tr>
<th>26</th>
<th>Mary</th>
<th>50</th>
<th>Julie</th>
<th>180</th>
<th>Mark</th>
<th>44</th>
<th>Shadow</th>
<th>111</th>
<th>Rachel</th>
<th>4</th>
<th>Pixel</th>
<th>95</th>
<th>Sue</th>
<th>196</th>
<th>James</th>
<th>170</th>
<th>Alex</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<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>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- 0-19
- 20-39
- 40-59
- 60-79
- 80-99
- 100-119
- 120-139
- 140-159
- 160-179
- 180-199

05-23: **Bucket Sort Example**
### 05-24: Bucket Sort Example

|  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | 50 | 180 | 44 | 111 | 4 | 95 | 196 | 170 |  |
|  | julie | mark | shadow | rachel | pixel | sue | james | alex | key |
|  |  |  |  |  |  |  |  |  | data |

0-19 40-59 80-99 120-139 160-179
20-39 60-79 100-119 140-159 180-199

05-25: Bucket Sort Example

|  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | 180 | 44 | 111 | 4 | 95 | 196 | 170 |  |  |
|  | mark | shadow | rachel | pixel | sue | james | alex |  | key |
|  |  |  |  |  |  |  |  |  | data |

0-19 40-59 80-99 120-139 160-179
20-39 60-79 100-119 140-159 180-199
05-26: Bucket Sort Example

05-27: Bucket Sort Example
### Bucket Sort Example

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>pixel</td>
</tr>
<tr>
<td>95</td>
<td>sue</td>
</tr>
<tr>
<td>196</td>
<td>james</td>
</tr>
<tr>
<td>170</td>
<td>alex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>julie</td>
</tr>
<tr>
<td>114</td>
<td>john</td>
</tr>
<tr>
<td>26</td>
<td>mary</td>
</tr>
<tr>
<td>44</td>
<td>shadow</td>
</tr>
<tr>
<td>111</td>
<td>rachel</td>
</tr>
<tr>
<td>180</td>
<td>mark</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-19</td>
<td>pixel</td>
<td>sue</td>
</tr>
<tr>
<td>20-39</td>
<td>44</td>
<td>john</td>
</tr>
<tr>
<td>40-59</td>
<td>26</td>
<td>mary</td>
</tr>
<tr>
<td>60-79</td>
<td>114</td>
<td>rachel</td>
</tr>
<tr>
<td>80-99</td>
<td>4</td>
<td>shadow</td>
</tr>
<tr>
<td>100-119</td>
<td>111</td>
<td>mark</td>
</tr>
<tr>
<td>120-139</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>140-159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160-179</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180-199</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
05-30: Bucket Sort Example

05-31: Bucket Sort Example
05-32: **Bucket Sort Example**

- Assume for the moment the keys are in the range 0..1, and are evenly distributed over the range

Bucket-Sort(A)
- \( n \leftarrow \text{length}[A] \)
- for \( i \leftarrow 1 \) to \( n \) do
  - append \( A[i] \) in list \( B[\lfloor n \times A[i] \rfloor] \)
- for \( i \leftarrow 0 \) to \( n-1 \) do
  - sort list \( B[i] \) with insertion sort
- Concatenate lists \( B[0] \ldots B[n] \)
05-34: **Bucket Sort Analysis**

- If we ignore the time for insertion sorts, algorithm takes time $\Theta(n)$
- Each insertion sort takes time $O(n^2)$, where $n_i$ is the length of list $n_i$

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

- Of course, $T(n)$ depends upon the lengths $n_i$

05-35: **Bucket Sort Analysis**

$$E[T(n)] = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

How can we calculate $E[n_i^2]$?

- Let $X_{ij} = I\{A[j] \text{ falls in bucket } i\}$

05-36: **Bucket Sort Analysis**

$$n_i = \sum_{j=1}^{n} X_{ij}$$

$$E[n_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= E\left[\sum_{j=1}^{n} \sum_{k=1}^{n} X_{ij} X_{ik}\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n, k \neq j} X_{ij} X_{ik}\right]$$

$$= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n, k \neq j} E[X_{ij} X_{ik}]$$

05-37: **Bucket Sort Analysis**

Calculating $E[(X_{ij})^2]$

- $X_{ij} = 1$ with probability $1/n$, and 0 with probability $1 - 1/n$
- Thus, $(X_{ij})^2 = 1$ with probability $1/n$ and 0 with probability $1 - 1/n$
And so: $E[(X_{ij})^2] = 1/n$

What about $E[X_{ij}X_{ik}]$?

**05-38: Bucket Sort Analysis**

Calculating $E[X_{ij}X_{ik}]$

- We assume that each element has an equal chance of landing in each bucket – so $E[X_{ij}] = 1/n$ and $E[X_{ik}] = 1/n$
- $X_{ij}$ and $X_{ik}$ are independent, so:

$$E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}] = \left(\frac{1}{n}\right)\left(\frac{1}{n}\right)$$

- We can now substitute the values $E[(X_{ij})^2] = 1/n$ and $E[X_{ij}X_{ik}] = 1/n^2$ into our equation for $E[(n_i)^2]$

**05-39: Bucket Sort Analysis**

$$E[n_i^2] = \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n, k \neq j} E[X_{ij}X_{ik}]$$

$$= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n, k \neq j} \frac{1}{n^2}$$

$$= n\left(\frac{1}{n}\right) + n(n-1)\frac{1}{n^2}$$

$$= 1 + \frac{n-1}{n}$$

$$= 2 - \frac{1}{n}$$

Finally, we can substitute back into the original equation for $T(n)$

**05-40: Bucket Sort Analysis**

$$E[T(n)] = E \left[ \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(1)$$

$$= \Theta(n)$$

**05-41: Counting Sort Revisited**

- Create the array $C[]$, such that $C[i] = \# \text{ of times key } i \text{ appears in the array.}$
Modify $C[i]$ such that $C[i]$ = the index of key $i$ in the sorted array. (assume no duplicate keys, for now)

If $x \not\in A$, we don’t care about $C[x]$

05-42: Counting Sort Revisited

- Create the array $C[\cdot]$, such that $C[i]$ = # of times key $i$ appears in the array.
- Modify $C[\cdot]$ such that $C[i]$ = the index of key $i$ in the sorted array. (assume no duplicate keys, for now)
- If $x \not\in A$, we don’t care about $C[x]$

```java
for (i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];
```

• Example: 3 1 2 4 9 8 7

05-43: Counting Sort Revisited

Once we have a modified $C$, such that $C[i]$ = index of key $i$ in the array, how can we use $C$ to sort the array?

```java
for (i=0; i<A.length; i++)
    B[C[A[i].key()]] = A[i];
for (i=0; i<A.length; i++)
    A[i] = B[i];
```

• Example: 3 1 2 4 9 8 7

05-44: Counting Sort Revisited

Once we have a modified $C$, such that $C[i]$ = index of key $i$ in the array, how can we use $C$ to sort the array?

```java
for (i=0; i<A.length; i++)
    C[A[i].key()]++;
for (i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];
```

What will the value of $C[i]$ represent?

05-45: Counting Sort & Duplicates

If a list has duplicate elements, and we create $C$ as before:

```java
for (i=0; i<A.length; i++)
    C[A[i].key()]++;
for (i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];
```

What will the value of $C[i]$ represent?

05-46: Counting Sort & Duplicates

If a list has duplicate elements, and we create $C$ as before:

```java
for (i=0; i<A.length; i++)
    C[A[i].key()]++;
for (i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];
```

What will the value of $C[i]$ represent?
• The last index in $A$ where element $i$ could appear.

05-47: (Almost) Final Counting Sort

```java
for(i=0; i<A.length; i++)
    C[A[i].key()]++;  // Initialize count array
for(i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];  // Accumulate counts

for (i=0; i<A.length; i++) {
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;
}
```

for (i=0; i<A.length; i++)
    A[i] = B[i];

• Example: 3 1 2 4 2 2 9 1 6

05-48: (Almost) Final Counting Sort

```java
for(i=0; i<A.length; i++)
    C[A[i].key()]++;  // Initialize count array
for(i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];  // Accumulate counts

for (i=0; i<A.length; i++) {
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;
}
```

for (i=0; i<A.length; i++)
    A[i] = B[i];

• Example: 3 1 2 4 2 2 9 1 6

• Is this a Stable sorting algorithm?

05-49: Final (!) Counting Sort

```java
for(i=0; i<A.length; i++)
    C[A[i].key()]++;  // Initialize count array
for(i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];  // Accumulate counts

for (i=A.length - 1; i>=0; i--) {  // From high to low key
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;
}
```

for (i=0; i<A.length; i++)
    A[i] = B[i];

05-50: Radix Sort
• Sort a list of numbers one digit at a time
  • Sort by 1st digit, then 2nd digit, etc
  • Each sort can be done in linear time, using counting sort

• First Try: Sort by most significant digit, then the next most significant digit, and so on
  • Need to keep track of a lot of sublists

05-51: **Radix Sort**  Second Try:
  • Sort by least significant digit first
  • Then sort by next-least significant digit, using a Stable sort
    
  • Sort by most significant digit, using a Stable sort

At the end, the list will be completely sorted. Why?

05-52: **Radix Sort**
  • If (most significant digit of $x$) $\neq$ (most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.

05-53: **Radix Sort**
  • If (most significant digit of $x$) $\neq$ (most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.
    • Last sort was by the most significant digit

05-54: **Radix Sort**
  • If (most significant digit of $x$) $\neq$ (most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.
    • Last sort was by the most significant digit
  • If (most significant digit of $x$) = (most significant digit of $y$) and
    (second most significant digit of $x$) $\neq$ (second most significant digit of $y$),
    then $x$ will appear in $A$ before $y$. 
05-55: **Radix Sort**

- If (most significant digit of \(x\)) \(\neq\) (most significant digit of \(y\)),
  then \(x\) will appear in \(A\) before \(y\).
  - Last sort was by the most significant digit
- If (most significant digit of \(x\)) = (most significant digit of \(y\)) and
  (second most significant digit of \(x\)) \(\neq\) (second most significant digit of \(y\)),
  then \(x\) will appear in \(A\) before \(y\).
  - After next-to-last sort, \(x\) is before \(y\). Last sort does not change relative order of \(x\) and \(y\)

### Original List

- 982 414 357 495 500 904 645 777 716 637 149 913 817 493 730 331 201

### Sorted by Least Significant Digit

- 500 730 331 201 982 493 913 414 904 645 495 716 357 777 637 817 149

### Sorted by Second Least Significant Digit

- 500 201 904 913 414 716 817 730 331 637 645 149 357 777 982 493 495

### Sorted by Most Significant Digit

- 149 201 331 357 414 493 495 500 637 645 716 730 777 817 904 913 982

05-56: **Radix Sort**

### Original List

- 982 437 649 500 902 433 167 842 803 500 637 645 716 730 331 201

### Sorted by Least Significant Base-100 Digit (last 2 base-10 digits)

- 982 437 649 500 902 433 167 842 803 500 637 645 716 730 331 201

### Sorted by Second Least Significant Base-100 Digit (first 2 base-10 digits)

- 1055 1673 2338 2493 3312 4333 4376 4409 6061 7004 8035 8442 1055 6061 1673 4376 2493

05-57: **Radix Sort**

- We do not need to use a single digit of the key for each of our counting sorts
  - We could use 2-digit chunks of the key instead
  - Our \(C\) array for each counting sort would have 100 elements instead of 10

### Original List

- 9823 4376 2493 1055 8502 4333 1673 8442 8035 6061 7004 3312 4409 2338

### Sorted by Least Significant Base-100 Digit (last 2 base-10 digits)

- 8502 7004 4409 3312 9823 4333 8035 2338 8442 1055 6061 1673 4376 2493

### Sorted by Most Significant Base-100 Digit (first 2 base-10 digits)

- 1055 1673 2338 2493 3312 4333 4376 4409 6061 7004 8035 8442 8502 9823
• “Digit” does not need to be base ten

• For any value $r$:
  • Sort the list based on $(\text{key} \mod r)$
  • Sort the list based on $((\text{key} / r) \mod r)$
  • Sort the list based on $((\text{key} / r^2) \mod r)$
  • Sort the list based on $((\text{key} / r^3) \mod r)$
  ...  
  • Sort the list based on $((\text{key} / r^{\log_k(\text{largest value in array})}) \mod r)$

• Code on other screen