Lecture 8
Counting sort, Bucket sort, Radix sort

CS 161 Design and Analysis of Algorithms
Ioannis Panageas
Counting sort

Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.

Warning: this algorithm description assumes that the arrays are indexed starting from 1, not from 0.

To implement the algorithm in a modern programming language directly from the algorithm description:

1. Allocate each array to be one entry larger than it actually is
2. Ignore location 0.

Main idea of CountingSort: Suppose $A$ contains exactly $j$ elements $\leq x$

1. If $x$ only appears once in $A$, then $x$ should go in $B[j]$.
2. If $x$ appears more than once in $A$ and we want a stable sort:
   1. Last occurrence of $x$ in $A$ should go in $B[j]$
   2. Next-to-last occurrence of $x$ should go in $B[j-1]$
   etc.
Counting sort

Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.

- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
    - Ignore location 0.
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
    - Ignore location 0.
- **Main idea of CountingSort:** Suppose $A$ contains exactly $j$ elements $\leq x$
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.

- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
    - Ignore location 0.

- **Main idea of CountingSort:** Suppose $A$ contains exactly $j$ elements $\leq x$
  - If $x$ only appears once in $A$, then $x$ should go in in $B[j]$. 

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.
- **Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
    - Ignore location 0.
- **Main idea of CountingSort:** Suppose $A$ contains exactly $j$ elements $\leq x$
  - If $x$ only appears once in $A$, then $x$ should go in in $B[j]$.
  - If $x$ appears more than once in $A$ and we want a stable sort:
Counting sort

- Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.

- **Warning**: this algorithm description assumes that the arrays are indexed starting from 1, not from 0.
  - To implement the algorithm in a modern programming language directly from the algorithm description:
    - Allocate each array to be one entry larger than it actually is
    - Ignore location 0.

- **Main idea of CountingSort**: Suppose $A$ contains exactly $j$ elements $\leq x$
  - If $x$ only appears once in $A$, then $x$ should go in $B[j]$.
  - If $x$ appears more than once in $A$ and we want a stable sort:
    - Last occurrence of $x$ in $A$ should go in $B[j]$. 

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting sort

Let $A$ be the input array, $B$ the output array. Assume there are $n$ items.

**Warning:** this algorithm description assumes that the arrays are indexed starting from 1, not from 0.

To implement the algorithm in a modern programming language directly from the algorithm description:

- Allocate each array to be one entry larger than it actually is
- Ignore location 0.

**Main idea of CountingSort:** Suppose $A$ contains exactly $j$ elements $\leq x$

- If $x$ only appears once in $A$, then $x$ should go in in $B[j]$.
- If $x$ appears more than once in $A$ and we want a stable sort:
  - Last occurrence of $x$ in $A$ should go in $B[j]$
  - Next-to-last occurrence of $x$ should go in $B[j - 1]$
  - etc.
Counting sort

Assume:
- We are sorting an array $A[1..n]$ of integers
- Each integer is in the range $1..k$
- Output array is $B[1..n]$
- Use an auxiliary array $locator[1..k]$
- $locator[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
- We make several passes over the data to set the values in the locator array before we do the actual sort.
- At the start of the final (sorting) pass, $locator[x]$ contains the number of elements $\leq x$
- On the final pass:
  - Process the input array $A$ from right to left. This makes the counting sort a stable sorting algorithm.
  - When a value of $x$ is encountered in the input array $A$:
    - Copy the value into location $locator[x]$ in the output array. That is, store it in location $B[locator[x]]$.
    - Decrement $locator[x]$. 
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers.
  - Each integer is in the range $1..k$.
  - Output array is $B[1..n]$.
  - Use an auxiliary array $locator[1..k]$.
    - $locator[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
    - We make several passes over the data to set the values in the $locator$ array before we do the actual sort.
    - At the start of the final (sorting) pass, $locator[x]$ contains the number of elements $\leq x$.
    - On the final pass:
      - Process the input array $A$ from right to left. This makes the counting sort a stable sorting algorithm.
      - When a value of $x$ is encountered in the input array $A$:
        - Copy the value into location $locator[x]$ in the output array. That is, store it in location $B[locator[x]]$.
        - Decrement $locator[x]$. 

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
  - Each integer is in the range $1..k$
Counting sort

Assume:

- We are sorting an array \( A[1..n] \) of integers
- Each integer is in the range \( 1..k \)
- Output array is \( B[1..n] \)
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
  - Each integer is in the range $1..k$
  - Output array is $B[1..n]$

- Use an auxiliary array $\text{locator}[1..k]$
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
  - Each integer is in the range $1..k$
  - Output array is $B[1..n]$

- Use an auxiliary array $locator[1..k]$
- $locator[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
Counting sort

▶ Assume:
  ▶ We are sorting an array $A[1..n]$ of integers
  ▶ Each integer is in the range $1..k$
  ▶ Output array is $B[1..n]$

▶ Use an auxiliary array $\text{locator}[1..k]$
▶ $\text{locator}[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
  ▶ We make several passes over the data to set the values in the locator array before we do the actual sort.
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
  - Each integer is in the range $1..k$
  - Output array is $B[1..n]$
- Use an auxiliary array $locator[1..k]$
- $locator[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
  - We make several passes over the data to set the values in the locator array before we do the actual sort.
  - At the start of the final (sorting) pass, $locator[x]$ contains the number of elements $\leq x$
Counting sort

▷ Assume:
  ▷ We are sorting an array $A[1..n]$ of integers
  ▷ Each integer is in the range $1..k$
  ▷ Output array is $B[1..n]$

▷ Use an auxiliary array $\text{locator}[1..k]$

▷ $\text{locator}[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
  ▷ We make several passes over the data to set the values in the locator array before we do the actual sort.
  ▷ At the start of the final (sorting) pass, $\text{locator}[x]$ contains the number of elements $\leq x$

▷ On the final pass:
Counting sort

- Assume:
  - We are sorting an array \( A[1..n] \) of integers
  - Each integer is in the range \( 1..k \)
  - Output array is \( B[1..n] \)
- Use an auxiliary array \( \text{locator}[1..k] \)
- \( \text{locator}[x] \) contains the index of the position in the output array \( B \) where a key of \( x \) should be stored.
  - We make several passes over the data to set the values in the locator array before we do the actual sort.
  - At the start of the final (sorting) pass, \( \text{locator}[x] \) contains the number of elements \( \leq x \)
- On the final pass:
  - Process the input array \( A \) from right to left (!). This makes the counting sort a stable sorting algorithm.
Counting sort

- Assume:
  - We are sorting an array $A[1..n]$ of integers
  - Each integer is in the range $1..k$
  - Output array is $B[1..n]$

- Use an auxiliary array $locator[1..k]$
  - $locator[x]$ contains the index of the position in the output array $B$ where a key of $x$ should be stored.
    - We make several passes over the data to set the values in the locator array before we do the actual sort.
    - At the start of the final (sorting) pass, $locator[x]$ contains the number of elements $\leq x$

- On the final pass:
  - Process the input array $A$ from right to left (!). This makes the counting sort a stable sorting algorithm.
  - When a value of $x$ is encountered in the input array $A$:
Counting sort

▶ Assume:
  ▶ We are sorting an array \( A[1..n] \) of integers
  ▶ Each integer is in the range \( 1..k \)
  ▶ Output array is \( B[1..n] \)

▶ Use an auxiliary array \( \text{locator}[1..k] \)
▶ \( \text{locator}[x] \) contains the index of the position in the output array \( B \) where a key of \( x \) should be stored.
  ▶ We make several passes over the data to set the values in the locator array before we do the actual sort.
  ▶ At the start of the final (sorting) pass, \( \text{locator}[x] \) contains the number of elements \( \leq x \)

▶ On the final pass:
  ▶ Process the input array \( A \) from right to left (!). This makes the counting sort a stable sorting algorithm.
  ▶ When a value of \( x \) is encountered in the input array \( A \):
    ▶ Copy the value into location \( \text{locator}[x] \) in the output array. That is, store it in location \( B[\text{locator}[x]] \)
Counting sort

- Assume:
  - We are sorting an array \( A[1..n] \) of integers
  - Each integer is in the range \( 1..k \)
  - Output array is \( B[1..n] \)

- Use an auxiliary array \( \text{locator}[1..k] \)
  - \( \text{locator}[x] \) contains the index of the position in the output array \( B \) where a key of \( x \) should be stored.
    - We make several passes over the data to set the values in the locator array before we do the actual sort.
    - At the start of the final (sorting) pass, \( \text{locator}[x] \) contains the number of elements \( \leq x \)

- On the final pass:
  - Process the input array \( A \) from right to left (!). This makes the counting sort a stable sorting algorithm.
  - When a value of \( x \) is encountered in the input array \( A \):
    - Copy the value into location \( \text{locator}[x] \) in the output array.
    - That is, store it in location \( B[\text{locator}[x]] \)
    - Decrement \( \text{locator}[x] \)
def CountingSort(A, B, n, k):
    //Initialize: set each locator[x] to
    //the number of entries \leq x
    for x = 1 to k do locator[x] = 0
    for i = 1 to n do locator[A[i]] = locator[A[i]] + 1
    for x = 2 to k do
        locator[x] = locator[x] + locator[x-1]
    //Fill output array, updating locator values
    for i = n down to 1 do
        B[locator[A[i]]] = A[i]
        locator[A[i]] = locator[A[i]] - 1

Analysis:
O(n + k) running time.
def CountingSort(A, B, n, k):
    //Initialize: set each locator[x] to the number of entries \leq x
    for x = 1 to k do locator[x] = 0
    for i = 1 to n do locator[A[i]] = locator[A[i]] + 1
    for x = 2 to k do
        locator[x] = locator[x] + locator[x-1]
    //Fill output array, updating locator values
    for i = n down to 1 do
        B[locator[A[i]]] = A[i]
        locator[A[i]] = locator[A[i]] - 1

Analysis: $O(n + k)$ running time.
Counting Sort Example

A:

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 3 & 5 & 7 & 5 & 7 & 3 & 8 & 7 & 4
\end{array}
\]
Counting Sort Example

A:

<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>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<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>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

```
1  3  5  7  5  7  3  8  7  4
```

locator:

```
1  1  3  4  6  6  9  10
```

B:

```
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 3 | 4 | 6 | 6 | 9 | 10 |

B:

|   |   |   |   |   |   |   |   |   |   |
Counting Sort Example

A:

\[
\begin{array}{cccccccccc}
1 & 3 & 5 & 7 & 5 & 7 & 3 & 8 & 7 & 4 \\
\end{array}
\]

locator:

\[
\begin{array}{cccccccc}
1 & 1 & 3 & 4 & 6 & 6 & 9 & 10 \\
\end{array}
\]

B:

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]
## Counting Sort Example

### A:

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

### locator:

<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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### B:

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting Sort Example

A:

<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>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<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>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

<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>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

locator:

B:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 3 | 3 | 6 | 6 | 9 | 10 |

B:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
Counting Sort Example

A:

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 3 & 5 & 7 & 5 & 7 & 3 & 8 & 7 & 4 \\
\end{array}
\]

locator:

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 1 & 3 & 3 & 6 & 6 & 9 & 10 \\
\end{array}
\]

B:

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 & & & 4 & & & & & 7 \\
\end{array}
\]
Counting Sort Example

A:

```
1 3 5 7 5 7 3 8 7 4
```

locator:

```
1 1 3 3 6 6 8 10
```

B:

```
  4
  7
```
Counting Sort Example

A:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<table>
<thead>
<tr>
<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>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

B:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

1 3 5 7 5 7 3 8 7 4

locator:

1 1 3 3 6 6 8 10

B:

1 2 3 4 5 6 7 8 9 10

4
7
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 3 | 3 | 6 | 6 | 8 | 10 |

B:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

|   |   |   | 4 |   |   |   | 7 | 8 |   |
# Counting Sort Example

## A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

## locator:

| 1 | 1 | 3 | 3 | 6 | 6 | 8 | 9 |

## B:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 7 | 8 |
Counting Sort Example

A:

```
1 3 5 7 5 7 3 8 7 4
```

locator:

```
1 1 3 3 6 6 8 9
```

B:

```
4
7 8
```
Counting Sort Example

A:

locator:

B:
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 3 | 3 | 6 | 6 | 8 | 9 |

B:

|   |   |   | 3 | 4 |   |   | 7 | 8 |   |
## Counting Sort Example

**A:**

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

**locator:**

| 1 | 1 | 3 | 3 | 6 | 6 | 8 | 9 |

**B:**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

```plaintext
1 2
3
3
4
4 5 6 7 8
7
9
8
10
```

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting Sort Example

A:

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<table>
<thead>
<tr>
<th></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>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

B:

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 2 | 3 | 6 | 6 | 8 | 9 |

B:

| 3 | 4 | 7 | 8 |
Counting Sort Example

A:

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

locator:

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

B:

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Counting Sort Example

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A:</strong></td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></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><strong>locator:</strong></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B:</strong></td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

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

locator:

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

B:

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

CompSci 161—Spring 2022—© M. B. Dillencourt—University of California, Irvine
Counting Sort Example

A:

<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>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<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>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B:

<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></td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
## Counting Sort Example

### Array A:

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

### Locator:

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

### Array B:

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Counting Sort Example

A:

1 3 5 7 5 7 3 8 7 4

locator:

1 1 2 3 5 6 7 9

B:

3 4 5 7 7 8
Counting Sort Example

A:

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

locator:

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

B:

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & | & 3 & | & 5 & | & 7 & | & 5 & | & 7 & | & 3 & | & 8 & | & 7 & | & 4 \\
\end{array}
\]

locator:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & | & 1 & | & 2 & | & 3 & | & 5 & | & 6 & | & 7 & | & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
| & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | & | \\
\end{array}
\]

B:
Counting Sort Example

A:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

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

B:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

<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>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<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>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B:

<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></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

<table>
<thead>
<tr>
<th>A:</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>5</th>
<th>7</th>
<th>3</th>
<th>8</th>
<th>7</th>
<th>4</th>
</tr>
</thead>
</table>

locator:

| 1 | 1 | 2 | 3 | 5 | 6 | 6 | 9 |

| B: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 3 | 4 | 5 | 7 | 7 | 7 | 7 | 8 |
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 2 | 3 | 5 | 6 | 6 | 9 |

B:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 3 | 4 | 5 | 7 | 7 | 7 | 7 | 8 |
Counting Sort Example

A:

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<table>
<thead>
<tr>
<th></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></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

B:

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 2 | 3 | 4 | 6 | 6 | 9 |

B:

| 3 | 4 | 5 | 5 | 7 | 7 | 7 | 8 |
Counting Sort Example

A:

1  3  5  7  5  7  3  8  7  4

locator:

1  1  2  3  4  6  6  9

B:

3  4  5  5  7  7  7  8
Counting Sort Example

A:

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 3 & 5 & 7 & 5 & 7 & 3 & 8 & 7 & 4 \\
\end{array}
\]

locator:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 1 & 2 & 3 & 4 & 6 & 6 & 9 \\
\end{array}
\]

B:

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
3 & 4 & 5 & 5 & 7 & 7 & 7 & 8 \\
\end{array}
\]
Counting Sort Example

A:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

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

B:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

locator:

<table>
<thead>
<tr>
<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>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

B:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

locator:

| 1 | 1 | 1 | 3 | 4 | 6 | 6 | 9 |

B:

| 3 | 3 | 4 | 5 | 5 | 7 | 7 | 7 | 8 |
Counting Sort Example

A:

1 3 5 7 5 7 3 8 7 4

locator:

1 1 1 3 4 6 6 9

B:

3 3 4 5 5 7 7 7 8
Counting Sort Example

A:

\[
\begin{array}{cccccccccccc}
1 & 3 & 5 & 7 & 5 & 7 & 3 & 8 & 7 & 4 \\
\end{array}
\]

locator:

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 3 & 4 & 6 & 6 & 9 \\
\end{array}
\]

B:

\[
\begin{array}{cccccccccccc}
1 & 3 & 3 & 4 & 5 & 5 & 7 & 7 & 7 & 8 \\
\end{array}
\]
Counting Sort Example

A:

| 1 | 3 | 5 | 7 | 5 | 7 | 3 | 8 | 7 | 4 |

Locator:

| 0 | 1 | 1 | 3 | 4 | 6 | 6 | 9 |

B:

| 1 | 3 | 3 | 4 | 5 | 5 | 7 | 7 | 7 | 8 |
Counting Sort Example

A: Done!

 locator:

B:
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
  1. Distribute keys into buckets
  2. Sort keys in each bucket
  3. Combine buckets.

  Simplest approach is to divide the space of possible keys into equal sized buckets.

Typically use insertion sort in phase 2.
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
  1. Distribute keys into buckets
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
  1. Distribute keys into buckets
  2. Sort keys in each bucket
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
  1. Distribute keys into buckets
  2. Sort keys in each bucket
  3. Combine buckets.
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.
- Three phases:
  1. Distribute keys into buckets
  2. Sort keys in each bucket
  3. Combine buckets.
- Simplest approach is to divide the space of possible keys into equal sized buckets.
Bucket Sort

- Divide space of possible keys into contiguous subranges, or buckets.

- Three phases:
  1. Distribute keys into buckets
  2. Sort keys in each bucket
  3. Combine buckets.

- Simplest approach is to divide the space of possible keys into equal sized buckets.

- Typically use insertion sort in phase 2.
Bucket Sort Example

Sort the following keys in the range 0-999, using 10 equal-size buckets:

661 74 835 140 198 923 113 642 467 449

1. Distribute
   0: 74
   1: 140 198 113
   2:
   3:
   4: 467 449
   5:
   6: 661 642
   7:
   8: 835
   9: 923

2. Sort
   0: 74
   1: 113 140 198
   2:
   3:
   4: 449 467
   5:
   6: 642 661
   7:
   8: 835
   9: 923

3. Combine
   74 113 140 198 449 467 642 661 835 923
Bucket Sort Example

Sort the following keys in the range 0-999, using 10 equal-size buckets:

661 74 835 140 198 923 113 642 467 449
Bucket Sort Example

Sort the following keys in the range 0-999, using 10 equal-size buckets:

661 74 835 140 198 923 113 642 467 449

1. Distribute

0: 74
1: 140 198 113
2:
3:
4: 467 449
5:
6: 661 642
7:
8: 835
9: 923
# Bucket Sort Example

Sort the following keys in the range 0-999, using 10 equal-size buckets:

\[661\ 74\ 835\ 140\ 198\ 923\ 113\ 642\ 467\ 449\]

1. Distribute

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>140 198 113</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>467 449</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>661 642</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>835</td>
</tr>
<tr>
<td>9</td>
<td>923</td>
</tr>
</tbody>
</table>

2. Sort

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>113 140 198</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>449 467</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>642 661</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>835</td>
</tr>
<tr>
<td>9</td>
<td>923</td>
</tr>
</tbody>
</table>
# Bucket Sort Example

Sort the following keys in the range 0-999, using 10 equal-size buckets:

661 74 835 140 198 923 113 642 467 449

## 1. Distribute

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>140, 198, 113</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>467, 449</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>661, 642</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>835</td>
</tr>
<tr>
<td>9</td>
<td>923</td>
</tr>
</tbody>
</table>

## 2. Sort

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>113, 140, 198</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>449, 467</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>642, 661</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>835</td>
</tr>
<tr>
<td>9</td>
<td>923</td>
</tr>
</tbody>
</table>

## 3. Combine

<table>
<thead>
<tr>
<th>Elements</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
</tr>
<tr>
<td>198</td>
<td></td>
</tr>
<tr>
<td>449</td>
<td></td>
</tr>
<tr>
<td>467</td>
<td></td>
</tr>
<tr>
<td>642</td>
<td></td>
</tr>
<tr>
<td>661</td>
<td></td>
</tr>
<tr>
<td>835</td>
<td></td>
</tr>
<tr>
<td>923</td>
<td></td>
</tr>
</tbody>
</table>

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Analysis of Bucket Sort

- **n** = number of items to sort
- **b** = number of buckets
- **s_i** = number of items in bucket i (i = 0, ..., b - 1)

Phase Running time:
1. Distribution: \( O(n) \)
2. Sorting each bucket: \( O(b + \sum_{i=1}^{b-1} s_i^2) \)
3. Combining buckets: \( O(b) \)

Total running time is: \( O(n + b + \sum_{i=1}^{b-1} s_i^2) \)

- **Worst case:** \( O(n^2) \)
- **Best case:** \( O(n) \)
- **Average case:** \( O(n) \) if certain assumptions are satisfied (next slide)

- Storage: \( O(n + b) \)
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]

Phase Running time

1. Distribution \( O(n) \)
2. Sorting each bucket \( O(b + \sum_{i=1}^{b-1} s_i^2) \)
3. Combining buckets \( O(b) \)

Total running time is: \( O(n + b + b \sum_{i=1}^{b-1} s_i^2) \)

▶ Worst case: \( O(n^2) \).
▶ Best case: \( O(n) \).
▶ Average case: \( O(n) \) if certain assumptions are satisfied (next slide)

▶ Storage: is \( O(n + b) \).

3-48
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]
\[ s_i = \text{number of items in bucket } i \ (i = 0, \ldots, b - 1) \]
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]
\[ s_i = \text{number of items in bucket } i \ (i = 0, \ldots, b - 1) \]

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
</table>

\[ \text{Total running time is: } O(n + b + \sum_{i=1}^{b-1} s_i^2) \]

▶ Worst case: \( O(n^2) \).
▶ Best case: \( O(n) \).
▶ Average case: \( O(n) \) if certain assumptions are satisfied (next slide).

Storage: \( O(n + b) \).
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]
\[ s_i = \text{number of items in bucket } i \ (i = 0, \ldots, b - 1) \]

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>
## Analysis of Bucket Sort

- $n$ = number of items to sort
- $b$ = number of buckets
- $s_i$ = number of items in bucket $i$ ($i = 0, \ldots, b - 1$)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>$O(b + \sum_i s_i^2)$</td>
</tr>
</tbody>
</table>
### Analysis of Bucket Sort

- \( n \) = number of items to sort
- \( b \) = number of buckets
- \( s_i \) = number of items in bucket \( i \) \((i = 0, \ldots, b - 1)\)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is: \( O(n + b + b \sum_i s_i^2) \)

- **Worst case:** \( O(n^2) \).
- **Best case:** \( O(n) \).
- **Average case:** \( O(n) \) if certain assumptions are satisfied (next slide)

Storage: \( O(n + b) \).
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]
\[ s_i = \text{number of items in bucket } i \ (i = 0, \ldots, b - 1) \]

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is:

\[ O \left( n + b + \sum_{i=1}^{b} s_i^2 \right) \]
Analysis of Bucket Sort

\( n = \) number of items to sort
\( b = \) number of buckets
\( s_i = \) number of items in bucket \( i \) (\( i = 0, \ldots, b - 1 \))

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is:

\[
O\left(n + b + \sum_{i=1}^{b} s_i^2\right)
\]

- Worst case: \( O(n^2) \).
Analysis of Bucket Sort

\[ n = \text{number of items to sort} \]
\[ b = \text{number of buckets} \]
\[ s_i = \text{number of items in bucket } i \ (i = 0, \ldots, b - 1) \]

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is:

\[
O \left( n + b + \sum_{i=1}^{b} s_i^2 \right)
\]

- **Worst case:** \( O(n^2) \).
- **Best case:** \( O(n) \).
Analysis of Bucket Sort

\( n = \) number of items to sort  
\( b = \) number of buckets  
\( s_i = \) number of items in bucket \( i \) \((i = 0, \ldots, b - 1)\)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is:

\[
O\left( n + b + \sum_{i=1}^{b} s_i^2 \right)
\]

- **Worst case**: \( O(n^2) \).
- **Best case**: \( O(n) \).
- **Average case**: \( O(n) \) if certain assumptions are satisfied (next slide)
## Analysis of Bucket Sort

- \( n \) = number of items to sort
- \( b \) = number of buckets
- \( s_i \) = number of items in bucket \( i \) \((i = 0, \ldots, b-1)\)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Distribution</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>2. Sorting each bucket</td>
<td>( O(b + \sum_i s_i^2) )</td>
</tr>
<tr>
<td>3. Combining buckets</td>
<td>( O(b) )</td>
</tr>
</tbody>
</table>

Total running time is:

\[
O \left( n + b + \sum_{i=1}^{b} s_i^2 \right)
\]

- **Worst case**: \( O(n^2) \).
- **Best case**: \( O(n) \).
- **Average case**: \( O(n) \) if certain assumptions are satisfied (next slide)
- **Storage**: is \( O(n + b) \).
Average running time of Bucket Sort

The following result is proved in [CLRS]:

Assume:

1. The number of buckets is equal to the number of keys (i.e., if \( b = n \))
2. The keys are distributed independently and uniformly over the buckets

Then the expected total cost of the intra-bucket sorts is \( O(n) \).
Radix Sort

- Useful for sorting multi-field keys in lexicographic order.
- Lexicographic order means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called dictionary order.
- Examples:
  - Words in dictionaries: clown comes before dog, cat comes before clown, car comes before cat.
  - Dates: (year, month, day).
  - Multi-digit numbers: (3-digit numbers in this example) 293 represented as (2,9,3), 71 represented as (0,7,1).
Radix Sort

- Useful for sorting multi-field keys in **lexicographic order**
Radix Sort

- Useful for sorting multi-field keys in **lexicographic order**
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on.
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- Lexicographic order means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called dictionary order
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- Lexicographic order means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called dictionary order
- Examples:
Radix Sort

- Useful for sorting multi-field keys in *lexicographic order*
- *Lexicographic order* means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called *dictionary order*
- **Examples:**
  - Words in dictionaries:
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
Radix Sort

▶ Useful for sorting multi-field keys in lexicographic order
▶ Lexicographic order means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called dictionary order
▶ Examples:
  ▶ Words in dictionaries:
    ▶ clown comes before dog
    ▶ cat comes before clown
Radix Sort

- Useful for sorting multi-field keys in **lexicographic order**
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
    - cat comes before clown
    - car comes before cat
  - Dates: (year, month, day)
  - Multi-digit numbers: (3-digit numbers in this example)
    - 293 represented as (2, 9, 3)
    - 71 represented as (0, 7, 1)
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
    - cat comes before clown
    - car comes before cat
  - Dates: (year, month, day)
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
    - cat comes before clown
    - car comes before cat
  - Dates: \((\text{year}, \text{month}, \text{day})\)
  - Multi-digit numbers: (3-digit numbers in this example)
Radix Sort

- Useful for sorting multi-field keys in **lexicographic order**
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
    - cat comes before clown
    - car comes before cat
  - Dates: (year, month, day)
  - Multi-digit numbers: (3-digit numbers in this example)
    - 293 represented as (2,9,3)
Radix Sort

- Useful for sorting multi-field keys in **lexicographic order**
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called **dictionary order**
- **Examples:**
  - Words in dictionaries:
    - *clown* comes before *dog*
    - *cat* comes before *clown*
    - *car* comes before *cat*
  - Dates: *(year, month, day)*
  - Multi-digit numbers: *(3-digit numbers in this example)*
    - 293 represented as *(2,9,3)*
    - 71 represented as *(0,7,1)*
Radix Sort

- Useful for sorting multi-field keys in lexicographic order
- **Lexicographic order** means sorted on the most important field, with ties broken on the next most important field, and so on. It is also called dictionary order
- **Examples:**
  - Words in dictionaries:
    - clown comes before dog
    - cat comes before clown
    - car comes before cat
  - Dates: (year, month, day)
  - Multi-digit numbers: (3-digit numbers in this example)
    - 293 represented as (2,9,3)
    - 71 represented as (0,7,1)
Radix Sort:

- Sorts on each field in the key, one at a time
- Sorts on the least-significant field first
- Uses a stable sort

Recall: A sorting algorithm is stable if whenever two keys are equal, the algorithm preserves their order (i.e., does not reverse them.)

```python
def radix_sort(A, n):
    for field in range(rightmost to leftmost):
        sort A on field using a stable sort
```
Radix Sort:

Radix sort:

Radix sort:

Recall: A sorting algorithm is stable if whenever two keys are equal, the algorithm preserves their order (i.e., does not reverse them.)

```python
def radix_sort(A, n):
    for field in range(rightmost to leftmost):
        sort A on field using a stable sort
```
Radix Sort:

Radix sort:

- Sorts on each field in the key, one at a time
Radix Sort:

Radix sort:
- Sorts on each field in the key, one at a time
- Sorts on least-significant field first
Radix Sort:

Radix sort:

- Sorts on each field in the key, one at a time
- Sorts on least-significant field first
- Uses a stable sort
Radix Sort:

Radix sort:

- Sorts on each field in the key, one at a time
- Sorts on least-significant field first
- Uses a stable sort
  - Recall: A sorting algorithm is stable if whenever two keys are equal, the algorithm preserves their order (i.e., does not reverse them.)
Radix Sort:

Radix sort:

- Sorts on each field in the key, one at a time
- Sorts on the least-significant field first
- Uses a stable sort
  - Recall: A sorting algorithm is stable if whenever two keys are equal, the algorithm preserves their order (i.e., does not reverse them.)

```python
def radix_sort(A, n):
    for field ranging from rightmost (least significant) to leftmost (most significant):
        sort A on field using a stable sort
```
Radix Sort:

Radix sort:

- Sorts on each field in the key, one at a time
- Sorts on the least-significant field first
- Uses a stable sort

Recall: A sorting algorithm is stable if whenever two keys are equal, the algorithm preserves their order (i.e., does not reverse them.)

```python
def radix_sort(A, n):
    for field ranging from rightmost (least significant)
        to leftmost (most significant):
            sort A on field using a stable sort
```
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field):

661 74 835 140 198 923 113 642 467 449

⇒ 140 661 642 923 113 074 835 467 198 449

⇒ 113 923 835 140 642 449 661 467 074 198

Note the importance of stability. We break the ties first, and stability makes sure the ties remain broken correctly.
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)

661 74 835 140 198 923 113 642 467 449

Note the importance of stability. We break the ties first, and stability makes sure the ties remain broken correctly.
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)

661  74  835  140  198  923  113  642  467  449

<table>
<thead>
<tr>
<th>661</th>
</tr>
</thead>
<tbody>
<tr>
<td>074</td>
</tr>
<tr>
<td>835</td>
</tr>
<tr>
<td>140</td>
</tr>
<tr>
<td>198</td>
</tr>
<tr>
<td>923</td>
</tr>
<tr>
<td>113</td>
</tr>
<tr>
<td>642</td>
</tr>
<tr>
<td>467</td>
</tr>
<tr>
<td>449</td>
</tr>
</tbody>
</table>
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)
661 74 835 140 198 923 113 642 467 449

\[
\begin{array}{cccccccc}
661 & 140 \\
074 & 661 \\
835 & 642 \\
140 & 923 \\
198 & 113 \\
923 & 074 \\
113 & 835 \\
642 & 467 \\
467 & 198 \\
449 & 449 \\
\end{array}
\]

Note the importance of stability.
We break the ties first, and stability makes sure the ties remain broken correctly.
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)

661 74 835 140 198 923 113 642 467 449

Note the importance of stability. We break the ties first, and stability makes sure the ties remain broken correctly.
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)
661 74 835 140 198 923 113 642 467 449

<table>
<thead>
<tr>
<th>661</th>
<th>140</th>
<th>113</th>
<th>074</th>
</tr>
</thead>
<tbody>
<tr>
<td>074</td>
<td>661</td>
<td>923</td>
<td>113</td>
</tr>
<tr>
<td>835</td>
<td>642</td>
<td>835</td>
<td>140</td>
</tr>
<tr>
<td>140</td>
<td>923</td>
<td>140</td>
<td>198</td>
</tr>
<tr>
<td>198</td>
<td>113</td>
<td>449</td>
<td>449</td>
</tr>
<tr>
<td>923</td>
<td>074</td>
<td>467</td>
<td>467</td>
</tr>
<tr>
<td>113</td>
<td>835</td>
<td>661</td>
<td>642</td>
</tr>
<tr>
<td>642</td>
<td>467</td>
<td>198</td>
<td>198</td>
</tr>
<tr>
<td>467</td>
<td>449</td>
<td>198</td>
<td>923</td>
</tr>
</tbody>
</table>
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)
661 74 835 140 198 923 113 642 467 449

Note the importance of stability.
Radix Sort Example:

Sort the following numbers using radix sort (each digit is a field)

661 74 835 140 198 923 113 642 467 449

<table>
<thead>
<tr>
<th>661</th>
<th>140</th>
<th>113</th>
<th>074</th>
</tr>
</thead>
<tbody>
<tr>
<td>074</td>
<td>661</td>
<td>923</td>
<td>113</td>
</tr>
<tr>
<td>835</td>
<td>642</td>
<td>835</td>
<td>140</td>
</tr>
<tr>
<td>140</td>
<td>923</td>
<td>140</td>
<td>198</td>
</tr>
<tr>
<td>198</td>
<td>113</td>
<td>449</td>
<td>449</td>
</tr>
<tr>
<td>923</td>
<td>074</td>
<td>467</td>
<td>467</td>
</tr>
<tr>
<td>113</td>
<td>835</td>
<td>661</td>
<td>661</td>
</tr>
<tr>
<td>642</td>
<td>467</td>
<td>198</td>
<td>198</td>
</tr>
<tr>
<td>467</td>
<td>198</td>
<td>074</td>
<td>923</td>
</tr>
<tr>
<td>449</td>
<td>449</td>
<td>198</td>
<td>923</td>
</tr>
</tbody>
</table>

Note the importance of stability.

We break the ties first, and stability makes sure the ties remain broken correctly.
Analysis of Radix Sort

Assume

▶ $n$ is the number of items

▶ $b$ is the size of each range

▶ Example:
  
  ▶ Each field of each item is a number in the range $0..b-1$.
  
  ▶ This is true if the numbers we are sorting are integers represented in base $b$.

▶ $d$ is the number of fields we are sorting

▶ For example, if each item is a base $b$ number with $d$ digits.

  (i.e., between $0$ and $b^d - 1$, inclusive).

▶ Each field is sorted using Bucket Sort or Counting Sort

Then the running time of radix sort is $O(d(n + b))$. 

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Analysis of Radix Sort

Assume
Analysis of Radix Sort

Assume

- $n$ is the number of items
Analysis of Radix Sort

Assume

- $n$ is the number of items
- $b$ is the size of each range

Each field of each item is a number in the range $0..b-1$. This is true if the numbers we are sorting are integers represented in base $b$. For example, if each item is a base $b$ number with $d$ digits (i.e., between $0$ and $b^d - 1$, inclusive).

Each field is sorted using Bucket Sort or Counting Sort. Then the running time of radix sort is $O(d(n + b))$. 

CompSci 161—Spring 2022—©M. B. Dillencourt—University of California, Irvine
Analysis of Radix Sort

Assume

- $n$ is the number of items
- $b$ is the size of each range
  - Example:
    - Each field of each item is a numbers in the range $0..b-1$. 

Analysis of Radix Sort

Assume

- $n$ is the number of items
- $b$ is the size of each range

**Example:**
- Each field of each item is a number in the range $0..b-1$.
- This is true if the numbers we are sorting are integers represented in base $b$. 
Analysis of Radix Sort

Assume

- $n$ is the number of items
- $b$ is the size of each range
  - Example:
    - Each field of each item is a number in the range $0..b-1$.
    - This is true if the numbers we are sorting are integers represented in base $b$.
- $d$ is the number of fields we are sorting
Analysis of Radix Sort

Assume

- \( n \) is the number of items
- \( b \) is the size of each range
  - Example:
    - Each field of each item is a numbers in the range \( 0..b-1 \).
    - This is true if the numbers we are sorting are integers represented in base \( b \).
- \( d \) is the number of fields we are sorting
  - For example, if each item is a base \( b \) number with \( d \) digits. (i.e., between 0 and \( b^d - 1 \), inclusive).
Analysis of Radix Sort

Assume

- $n$ is the number of items
- $b$ is the size of each range
  - Example:
    - Each field of each item is a numbers in the range $0..b-1$.
    - This is true if the numbers we are sorting are integers represented in base $b$.
- $d$ is the number of fields we are sorting
  - For example, if each item is a base $b$ number with $d$ digits.
    (i.e., between 0 and $b^d - 1$, inclusive).
- Each field is sorted using Bucket Sort or Counting Sort
Analysis of Radix Sort

Assume

- \( n \) is the number of items
- \( b \) is the size of each range
  - Example:
    - Each field of each item is a number in the range 0..\( b-1 \).
    - This is true if the numbers we are sorting are integers represented in base \( b \).
- \( d \) is the number of fields we are sorting
  - For example, if each item is a base \( b \) number with \( d \) digits. (i.e., between 0 and \( b^d - 1 \), inclusive).
  - Each field is sorted using Bucket Sort or Counting Sort
Then the running time of radix sort is \( O(d(n + b)) \).
Deterministic Selection:
Find k-th element
Recall QuickSelect

\[ \text{quickSelect}(S, k) \]

If \( n \) is small, brute force and return.
Pick a random \( x \in S \) and put rest into:
\[ L, \text{ elements smaller than } x \]
\[ G, \text{ elements greater than } x \]
\[ \text{if } k \leq |L| \text{ then} \]
\[ \text{quickSelect}(L, k) \]
\[ \text{else if } k == |L| + 1 \text{ then} \]
\[ \text{return } x \]
\[ \text{else} \]
\[ \text{quickSelect}(G, k - (|L| + 1)) \]
Deterministic Selection

Instead of picking $x$ at random:
- Divide $S$ into $g = \lceil n/5 \rceil$ groups
- Each group has 5 elements (except maybe $g^{th}$)
- Find median of each group of 5
- Find median of those medians
- Let $x$ be that median.

We call this the “medians of 5” method.
Selecting Median of 5 Example

<p>| | | | | | | | | |</p>
<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>870</td>
<td>647</td>
<td>845</td>
<td>742</td>
<td>372</td>
<td>882</td>
<td>691</td>
<td>341</td>
<td>461</td>
</tr>
<tr>
<td>989</td>
<td>151</td>
<td>100</td>
<td>729</td>
<td>101</td>
<td>397</td>
<td>825</td>
<td>587</td>
<td>363</td>
</tr>
<tr>
<td>595</td>
<td>524</td>
<td>930</td>
<td>259</td>
<td>133</td>
<td>955</td>
<td>620</td>
<td>970</td>
<td>430</td>
</tr>
<tr>
<td>839</td>
<td>139</td>
<td>735</td>
<td>590</td>
<td>782</td>
<td>913</td>
<td>378</td>
<td>474</td>
<td>255</td>
</tr>
<tr>
<td>875</td>
<td>150</td>
<td>791</td>
<td>779</td>
<td>792</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Deterministic Select

DeterministicSelect($S, k$)

If $n$ is small, brute force and return.
Pick $x \in S$ via medians-of-5 and put rest into:
- $L$, elements smaller than $x$
- $G$, elements greater than $x$

if $k \leq |L|$ then
    DeterministicSelect($L, k$)
else if $k == |L| + 1$ then
    return $x$
else
    DeterministicSelect($G, k - (|L| + 1)$)
Deterministic Selection

Let’s visualize: how does pivot compare to list?
Demo Re-visualized

- Each column was a group of five.
- Each column is sorted
- Columns are ordered based on median-of-5
- Which cells are in $L$? $G$? Either?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>283</td>
<td>255</td>
<td>133</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>363</td>
<td>378</td>
<td>259</td>
<td>461</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>397</td>
<td>474</td>
<td>524</td>
<td>596</td>
<td>620</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>735</td>
<td>742</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>791</td>
<td></td>
</tr>
<tr>
<td></td>
<td>691</td>
<td>955</td>
<td>782</td>
<td>845</td>
<td>792</td>
</tr>
<tr>
<td></td>
<td>882</td>
<td>970</td>
<td>839</td>
<td>870</td>
<td>875</td>
</tr>
</tbody>
</table>
Deterministic Selection

- How few elements *must be* smaller than pivot?

- How few *must be* non-smaller than pivot?

- How many could be in either group?