12-0: **Comparison Sorting**

- Comparison sorts work by comparing elements
  - Can only compare 2 elements at a time
  - Check for <, >, =.
- All the sorts we have seen so far (Insertion, Quick, Merge, Heap, etc.) are comparison sorts
- If we know nothing about the list to be sorted, we need to use a comparison sort

12-1: **Decision Trees**

Insertion Sort on list \{a, b, c\}

```
a < b < c   b < c < a
a < c < b   c < a < b
b < a < c   c < b < a
```

12-2: **Decision Trees**

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?

12-3: **Decision Trees**

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  - (The depth of the shallowest leaf) + 1
- What is the worst case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?

12-4: **Decision Trees**

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  - (The depth of the shallowest leaf) + 1
• What is the worst case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  • The height of the tree – (depth of the deepest leaf) + 1

12-5: Decision Trees

• What is the largest number of nodes for a tree of depth \( d \)?

12-6: Decision Trees

• What is the largest number of nodes for a tree of depth \( d \)?
  • \( 2^d \)

• What is the minimum height, for a tree that has \( n \) leaves?
  • \( \lg n \)

• How many leaves are there in a decision tree for sorting \( n \) elements?

12-7: Decision Trees

• What is the largest number of nodes for a tree of depth \( d \)?
  • \( 2^d \)

• What is the minimum height, for a tree that has \( n \) leaves?
  • \( \lg n \)

• How many leaves are there in a decision tree for sorting \( n \) elements?
  • \( n! \)

12-8: Decision Trees

• What is the largest number of nodes for a tree of depth \( d \)?
  • \( 2^d \)

• What is the minimum height, for a tree that has \( n \) leaves?
  • \( \lg n \)

• How many leaves are there in a decision tree for sorting \( n \) elements?
  • \( n! \)

• What is the minimum height, for a decision tree for sorting \( n \) elements?
• \( \lg n! \)

12-10: \( \lg(n!) \in \Omega(n \lg n) \)

\[
\begin{align*}
\lg(n!) &= \lg(n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 2 \cdot 1) \\
&= (\lg n) + (\lg(n - 1)) + (\lg(n - 2)) + \ldots + (\lg 2) + (\lg 1) \\
&\geq (\lg n) + (\lg(n - 1)) + \ldots + (\lg(n/2)) \\
&\geq (\lg(n/2) + (\lg(n/2)) + \ldots + (\lg(n/2)) \\
&= (n/2) \lg(n/2) \\
&\in \Omega(n \lg n)
\end{align*}
\]

12-11: **Sorting Lower Bound**

• All comparison sorting algorithms can be represented by a decision tree with \( n! \) leaves

• Worst-case number of comparisons required by a sorting algorithm represented by a decision tree is the height of the tree

• A decision tree with \( n! \) leaves must have a height of at least \( n \lg n \)

• All comparison sorting algorithms have worst-case running time \( \Omega(n \lg n) \)

12-12: **Counting Sort**

• Sorting a list of \( n \) integers

• We know all integers are in the range 0 . . . \( m \)

• We can potentially sort the integers faster than \( n \lg n \)

• 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

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

12-13: **Counting Sort Example**

```
3 1 3 5 2 1 6 7 8 1
0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
```
12-14: Counting Sort Example
135216781

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

12-15: Counting Sort Example
35216781

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

12-16: Counting Sort Example
5216781

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

12-17: Counting Sort Example
216781

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</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>

12-18: Counting Sort Example
6781

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</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>

12-19: Counting Sort Example
6781

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</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>

12-20: Counting Sort Example
12-21: Counting Sort Example

12-22: Counting Sort Example

12-23: Counting Sort Example

12-24: Counting Sort Example

12-25: $\Theta()$ of Counting Sort

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

12-26: $\Theta()$ 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 $\Omega(n \log n)$ bound for all sorting algorithms?
12-27: $\Theta()$ 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 $\Omega(n \log n)$ bound for all sorting algorithms?
  - For Comparison Sorts, which allow for sorting arbitrary data. What happens when $m$ is very large?

12-28: 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 $i$.

12-29: Binsort Example

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

12-30: Binsort Example

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

12-31: Binsort Example
12-32: **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.

12-33: **Bucket Sort Example**

<table>
<thead>
<tr>
<th>114</th>
<th>26</th>
<th>50</th>
<th>180</th>
<th>44</th>
<th>111</th>
<th>4</th>
<th>95</th>
<th>196</th>
<th>170</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>mary</td>
<td>julie</td>
<td>mark</td>
<td>shadow</td>
<td>rachel</td>
<td>pixel</td>
<td>sue</td>
<td>james</td>
<td>alex</td>
</tr>
</tbody>
</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 40-59 80-99 120-139 160-179
20-39 60-79 100-119 140-159 180-199

12-34: **Bucket Sort Example**
### Bucket Sort Example

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

- 114: john

<table>
<thead>
<tr>
<th>0-19</th>
<th>40-59</th>
<th>80-99</th>
<th>120-139</th>
<th>160-179</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-39</td>
<td>60-79</td>
<td>100-119</td>
<td>140-159</td>
<td>180-199</td>
</tr>
</tbody>
</table>

12-35: Bucket Sort Example

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

- 26: mary
- 114: john

<table>
<thead>
<tr>
<th>0-19</th>
<th>40-59</th>
<th>80-99</th>
<th>120-139</th>
<th>160-179</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-39</td>
<td>60-79</td>
<td>100-119</td>
<td>140-159</td>
<td>180-199</td>
</tr>
</tbody>
</table>

12-36: Bucket Sort Example
12-37: Bucket Sort Example

12-38: Bucket Sort Example
### 12-39: Bucket Sort Example

|------|-------|-------|-------|-------|---------|----------|----------|----------|----------|

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

### 12-40: Bucket Sort Example

|------|-------|-------|-------|-------|---------|----------|----------|----------|----------|

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>rachel</td>
</tr>
<tr>
<td>50</td>
<td>julie</td>
</tr>
<tr>
<td>26</td>
<td>mary</td>
</tr>
<tr>
<td>114</td>
<td>john</td>
</tr>
<tr>
<td>180</td>
<td>mark</td>
</tr>
</tbody>
</table>
12-41: Bucket Sort Example

12-42: Bucket Sort Example
### Bucket Sort Example

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-19</td>
<td></td>
</tr>
<tr>
<td>20-39</td>
<td></td>
</tr>
<tr>
<td>40-59</td>
<td></td>
</tr>
<tr>
<td>60-79</td>
<td></td>
</tr>
<tr>
<td>80-99</td>
<td></td>
</tr>
<tr>
<td>100-119</td>
<td></td>
</tr>
<tr>
<td>120-139</td>
<td></td>
</tr>
<tr>
<td>140-159</td>
<td></td>
</tr>
<tr>
<td>160-179</td>
<td></td>
</tr>
<tr>
<td>180-199</td>
<td></td>
</tr>
</tbody>
</table>

- **Key**: John, Julie, Mary, Sue, Rachel, Alex, James, Pixel, Sue, James, Pixel, Rachel, Alex, Mary
- **Data**: 50, 44, 95, 111, 170, 196, 114, 26, 180, 170

---

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-19</td>
<td></td>
</tr>
<tr>
<td>20-39</td>
<td></td>
</tr>
<tr>
<td>40-59</td>
<td></td>
</tr>
<tr>
<td>60-79</td>
<td></td>
</tr>
<tr>
<td>80-99</td>
<td></td>
</tr>
<tr>
<td>100-119</td>
<td></td>
</tr>
<tr>
<td>120-139</td>
<td></td>
</tr>
<tr>
<td>140-159</td>
<td></td>
</tr>
<tr>
<td>160-179</td>
<td></td>
</tr>
<tr>
<td>180-199</td>
<td></td>
</tr>
</tbody>
</table>

- **Key**: John, Julie, Mary, Sue, Rachel, Alex, James, Pixel, Sue, James, Pixel, Rachel, Alex, Mary
- **Data**: 50, 44, 95, 111, 170, 196, 114, 26, 180, 170
12-45: **Counting Sort Revisited**

- We’re going to look at counting sort again.
- For the moment, we will assume that our array is indexed from $1 \ldots n$ (where $n$ is the number of elements in the list) instead of being indexed from $0 \ldots n-1$, to make the algorithm easier to understand.
- Later, we will go back and change the algorithm to allow for an index between $0 \ldots n-1$.

12-46: **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 $C[i]$ is the *index* of key $i$ in the sorted array. (assume no duplicate keys, for now)
- If $x \notin A$, we don’t care about $C[x]$.

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

12-47: **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 $C[i]$ is the *index* of key $i$ in the sorted array. (assume no duplicate keys, for now)
- If $x \notin A$, we don’t care about $C[x]$.

- Example: 3 1 2 4 9 8 7

12-48: **Counting Sort Revisited**

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

12-49: **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?

```
for (i=1; i <= n; i++)
    B[C[A[i].key()]] = A[i];
```

```
for (i=1; i <= n; i++)
    A[i] = B[i];
```

• Example: 3 1 2 4 9 8 7

12-50: Counting Sort & Duplicates

• If a list has duplicate elements, and we create \( C \) as before:

```
for(i=1; i <= n; 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?

12-51: Counting Sort & Duplicates

• If a list has duplicate elements, and we create \( C \) as before:

```
for(i=1; i <= n; 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.

12-52: (Almost) Final Counting Sort

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

```
for (i=1; i <= n; i++) {
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;}
for (i=1; i <= n; i++)
    A[i] = B[i];
```

• Example: 3 1 2 4 2 2 9 1 6

12-53: (Almost) Final Counting Sort
for(i=1; i <= n; i++)
    C[A[i].key()]++;
for(i=1; i<C.length; i++)
    C[i] = C[i] + C[i-1];

for (i=1; i <= n; i++) {
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;
}
for (i=1; i <= n; i++)
    A[i] = B[i];

- Example: 3 1 2 4 2 2 9 1 6
- Is this a Stable sorting algorithm?

12-54: (Almost) Final Counting Sort

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

for (i = n; i>=1; i++) {
    B[C[A[i].key()]] = A[i];
    C[A[i].key()]--;
}
for (i=1; i < n; i++)
    A[i] = B[i];

- How would we change this algorithm if our arrays were indexed from 0...n - 1 instead of 1...n?

12-55: Final (!) Counting Sort

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];

for (i=A.length - 1; i>=0; i++) {
    C[A[i].key()]--;
    B[C[A[i].key()]] = A[i];
}
for (i=0; i < A.length; i++)
    A[i] = B[i];

12-56: 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

12-57: 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?

12-58: Radix Sort
  • If (most significant digit of $x$) $\lt$ (most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.

12-59: Radix Sort
  • If (most significant digit of $x$) $\lt$ (most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.
  • Last sort was by the most significant digit

12-60: Radix Sort
  • If (most significant digit of $x$) $\lt$
    (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 $x$) $\lt$
    (second most significant digit of $y$),
    then $x$ will appear in $A$ before $y$.

12-61: Radix Sort
- If (most significant digit of $x$) $\downarrow$
  (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 $x$) $\downarrow$
  (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 $y$

12-62: **Radix Sort**

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
```

12-63: **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

12-64: **Radix Sort**

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
```

12-65: **Radix Sort**

- “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
    $((\text{key} / r^{\log_4 \text{(largest value in array)}}) \mod r)$

• Code on other screen