5.1 Key-Indexed Counting Demo
**Goal.** Sort an array \(a[]\) of \(N\) integers between 0 and \(R - 1\).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array $a[]$ of $N$ integers between 0 and $R - 1$.
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];
for (int i = 0; i < N; i++)
    count[a[i]+1]++;
for (int r = 0; r < R; r++)
    count[r+1] += count[r];
for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];
for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
**Key-indexed counting**

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

<table>
<thead>
<tr>
<th>i</th>
<th>a[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td>f</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>6</td>
<td>d</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>f</td>
</tr>
<tr>
<td>9</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>e</td>
</tr>
<tr>
<td>11</td>
<td>a</td>
</tr>
</tbody>
</table>

6 keys < d, 8 keys < e
so d's go in a[6] and a[7]
Key-indexed counting

**Goal.** Sort an array \( a[i] \) of \( N \) integers between 0 and \( R - 1 \).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];
for (int i = 0; i < N; i++)
    count[a[i]+1]++;
for (int r = 0; r < R; r++)
    count[r+1] += count[r];
for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];
for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];
for (int i = 0; i < N; i++)
    count[a[i]+1]++;
for (int r = 0; r < R; r++)
    count[r+1] += count[r];
for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];
for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
**Goal.** Sort an array $a[]$ of $N$ integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array `a[]` of `N` integers between 0 and `R - 1`.
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];
for (int i = 0; i < N; i++)
    count[a[i]+1]++;
for (int r = 0; r < R; r++)
    count[r+1] += count[r];
for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];
for (int i = 0; i < N; i++)
a[i] = aux[i];
```
### Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- **Access cumulates using key as index to move items.**
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

<table>
<thead>
<tr>
<th>( i )</th>
<th>( a[i] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td>f</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>6</td>
<td>d</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>f</td>
</tr>
<tr>
<td>9</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>e</td>
</tr>
<tr>
<td>11</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( r )</th>
<th>count[( r )]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>d</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>f</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array $a[]$ of $N$ integers between 0 and $R-1$.
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];
for (int i = 0; i < N; i++)
    count[a[i]+1]++;
for (int r = 0; r < R; r++)
    count[r+1] += count[r];
for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];
for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array $a[]$ of $N$ integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
**Key-indexed counting**

**Goal.** Sort an array \( a[] \) of \( N \) integers between 0 and \( R - 1 \).
- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
**Goal.** Sort an array $a[]$ of $N$ integers between $0$ and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```
Key-indexed counting

**Goal.** Sort an array $a[]$ of $N$ integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- **Copy back into original array.**

```java
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```