THE COUNTING SORT

The counting sort is an efficient algorithm for sorting values that have a limited range. It was invented by Harold H. Seward in the mid 1950s.

Suppose you have an array \( v \) containing \( m \) integers, each within the range 0 to \( m-1 \), shuffled into random order. You can sort these integers simply by moving each integer into its correct position within an auxiliary array.

**Example**
The picture below shows how the first three items in array \( v \) are moved into their correct positions within the auxiliary array.

Here’s the counting sort algorithm:

```
Counting Sort Algorithm

int [] aux = new int[m];
for ( int k=0; k < m; k++ )
    aux[v[k]] = v[k];
```
A more realistic situation assumes that array $v$ contains $n$ integers in the range 0 to $m-1$, where $m$ is within some constant factor of $n$ (i.e. $m < cn$ for some constant $c > 0$). Also, duplicate values are allowed. Under these conditions, the counting sort works in three passes. The first pass counts each integer in $v$:

### Revised Counting Sort Algorithm

#### Pass 1

```java
int[] count = new int[m];
for (int k=0; k < n; k++)
    count[v[k]]++;
```

#### Example

<table>
<thead>
<tr>
<th></th>
<th>0</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>count</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each integer $k$ occupies $\text{count}[k]$ positions in the final sorted array. If integer $k$ starts at position $p$ then, for it to occupy $\text{count}[k]$, integer $k+1$ must start at position $p+\text{count}[k]$.

#### Example, continued

<table>
<thead>
<tr>
<th></th>
<th>0</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>count</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the sorted array, integer 0 starts at position 0 and occupies $\text{count}[0] = 6$ positions. Thus, integer 1 starts at position $0+6 = 6$. Likewise, integer 2 starts at position $6 + \text{count}[1] = 6 + 4 = 10$. 

---

The Counting Sort
The second pass of the algorithm calculates all of these starting positions and places them into a third array.

![Revised Counting Sort Algorithm]

```
int [] start = new int[m];
start[0] = 0;
for ( int k=1; k < m; k++ )
  start[k] = start[k-1] + count[k-1];
```

---

**Example, continued**

<table>
<thead>
<tr>
<th>v</th>
<th>0</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

| count | 6 | 4 | 2 | 3 |
|       |   |   |   |   |

| start | 0 | 6 | 10 | 12 |
The third and final pass distributes each integer in the original array \(v\) to its final position in the sorted array:

Revised Counting Sort Algorithm
Pass 3

```java
int[] fin = new int[n];
for (int k=0; k < n; k++)
    fin[start[v[k]]++] = v[k];
```

**Example, concluded**

<table>
<thead>
<tr>
<th></th>
<th>0</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v)</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>count</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<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>6</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fin</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Programming Exercises

1. Implement the revised counting sort algorithm as the following Java method:

   ```java
   int[] countingSort(int[] v, int m)
   // Use Seward's counting sort algorithm that
   // returns an array containing the items in 'v'
   // in ascending order.
   // Each v[k] is in the range 0 to m.
   ```

   Write an application to test your method. The application must create an unsorted array, call your method to sort it and print the results.