Consider the following 9 numbers:

493   812   715   710   195   437   582   340   385

We should start sorting by comparing and ordering the **one's** digits:

|  |  |
| --- | --- |
| **Digit** | **Sublist** |
| 0 | 340 710 |
| 1 |  |
| 2 | 812 582 |
| 3 | 493 |
| 4 |  |
| 5 | 715 195 385 |
| 6 |  |
| 7 | 437 |
| 8 |  |
| 9 |  |

Notice that the numbers were added onto the list in the order that they were found, which is why the numbers appear to be unsorted in each of the sublists above. Now, we gather the sublists (in order from the 0 sublist to the 9 sublist) into the main list again:

340   710   812   582   493   715   195   385   437

Note: The **order** in which we divide and reassemble the list is **extremely important**, as this is one of the foundations of this algorithm.

Now, the sublists are created again, this time based on the **ten's** digit:

|  |  |
| --- | --- |
| **Digit** | **Sublist** |
| 0 |  |
| 1 | 710 812 715 |
| 2 |  |
| 3 | 437 |
| 4 | 340 |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 | 582 385 |
| 9 | 493 195 |

Now the sublists are gathered in order from 0 to 9:

710   812   715   437   340   582   385   493   195

Finally, the sublists are created according to the **hundred's** digit:

|  |  |
| --- | --- |
| **Digit** | **Sublist** |
| 0 |  |
| 1 | 195 |
| 2 |  |
| 3 | 340 385 |
| 4 | 437 493 |
| 5 | 582 |
| 6 |  |
| 7 | 710 715 |
| 8 | 812 |
| 9 |  |

At last, the list is gathered up again:

195   340   385   437   493   582   710   715   812

And now we have a fully sorted array! Radix Sort is very simple, and a computer can do it fast. When it is programmed properly, Radix Sort is in fact **one of the fastest sorting algorithms** for numbers or strings of letters.