

Radix Sort

Comparison sort runtime of $O(n \cdot \log(n))$ is optimal

- The *problem* of sorting cannot be solved using comparisons with less than $n \cdot \log(n)$ time complexity
- See Proposition I in Chapter 2.2 of the text

How can we sort without comparison?

- Consider the following approach:
 - Look at the least-significant digit
 - Group numbers with the same digit
 - Maintain relative order
 - Place groups back in array together
 - I.e., all the 0's, all the 1's, all the 2's, etc.
- Repeat for increasingly significant digits

The characteristics of Radix sort

- Least significant digit (LSD) Radix sort
 - a fast stable sorting algorithm
 - begins at the least significant digit (e.g. the rightmost digit)
 - proceeds to the most significant digit (e.g. the leftmost digit)
 - lexicographic orderings

Generally Speaking

- 1. Take the least significant digit (or group of bits) of each key
- 2. Group the keys based on that digit, but otherwise keep the original order of keys.
 - This is what makes the LSD radix sort a stable sort.
- 3. Repeat the grouping process with each more significant digit

Generally Speaking

```
public static void sort(String [] a, int W) {  
    int N = a.length;  
    int R = 256;  
    String [] aux = new String[N];  
    for (int d = W - 1; d >= 0; --d) {  
        aux = sorted array a by the dth character  
        a = aux  
    }  
}
```

A Radix sort example

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

2IYE230
3CI0720
1ICK750
1ICK750
3CI0720
3ATW723
4JZY524
10HV845
10HV845
10HV845
4PGC938
2RLA629
2RLA629

3CI0720
3CI0720
3ATW723
4JZY524
2RLA629
2RLA629
2IYE230
4PGC938
10HV845
10HV845
10HV845
10HV845
1ICK750
1ICK750

2IYE230
4JZY524
2RLA629
2RLA629
3CI0720
3CI0720
3ATW723
1ICK750
1ICK750
10HV845
10HV845
10HV845
10HV845
4PGC938

3CI0720
3CI0720
2RLA629
2RLA629
4PGC938
2IYE230
1ICK750
1ICK750
10HV845
10HV845
10HV845
10HV845
3ATW723
4JZY524

A Radix sort example

3CI0720
3CI0720
2RLA629
2RLA629
4PGC938
2IYE230
1ICK750
1ICK750
10HV845
10HV845
10HV845
3ATW723
4JZY524

1ICK750
1ICK750
4PGC938
10HV845
10HV845
10HV845
3CI0720
3CI0720
2RLA629
2RLA629
3ATW723
2IYE230
4JZY524

3ATW723
3CI0720
3CI0720
1ICK750
1ICK750
2IYE230
4JZY524
10HV845
10HV845
10HV845
4PGC938
2RLA629
2RLA629

1ICK750
1ICK750
10HV845
10HV845
10HV845
2IYE230
2RLA629
2RLA629
3ATW723
3CI0720
3CI0720
4JZY524
4PGC938

A Radix sort example

- Problem: How to ?
 - Group the keys based on that digit,
 - but otherwise keep the original order of keys.

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

2IYE230
3CI0720
1ICK750
1ICK750
3CI0720
3ATW723
4JZY524
10HV845
10HV845
10HV845
4PGC938
2RLA629
2RLA629

Key-indexed counting

- 1. Take the least significant digit (or group of bits) of each key
- 2. Group the keys based on that digit, but otherwise keep the original order of keys.
 - This is what makes the LSD radix sort a stable sort.
- 3. Repeat the grouping process with each more significant digit
- The sort in step 2 is usually done using **bucket sort or counting sort**, which are efficient in this case

Key-indexed counting

```
public static void sort(String[] a, int W) {  
    int N = a.length;  
    int R = 256;  
    String[] aux = new String[N];  
    for (int d = W - 1; d >= 0; --d) {  
        aux = sorted array a by the dth character (stable  
sort)  
        a = aux  
    }  
}
```

Key-indexed counting

```
int [] count = new int[R + 1];  
// step1: calculate the histogram of key frequencies  
for (int i = 0; i < N; ++i) {  
    count[a[i].charAt(d) + 1]++;  
}  
// step 2: calculate the starting index for each key  
for (int r = 0; r < R; ++r) {  
    count[r + 1] += count[r];  
}  
// step 3: copy to output array, preserving order of with equal  
keys:  
for (int i = 0; i < N; ++i) {  
    aux[count[a[i].charAt(d)]++] = a[i];  
}  
for (int i = 0; i < N; ++i) {  
    a[i] = aux[i];  
}
```

Key-indexed counting

- *Ascii charset: 256 entries*

- *Digits charset as an example*
- *Sort by the right-most digit*

- *Step 1: count the frequencies*

0: 0
1: 5
2: 0
3: 0
4: 1
5: 1
6: 3
7: 0
8: 0
9: 1
10: 2

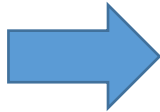
```
int [] count = new int[R + 1];  
// step1: calculate the histogram of key frequencies  
for (int i = 0; i < N; ++i) {  
    count[a[i].charAt(d) + 1]++;  
}  
// step 2: calculate the starting index for each key  
for (int r = 0; r < R; ++r) {  
    count[r + 1] += count[r];  
}  
// step 3: copy to output array, preserving order of with equal keys:  
for (int i = 0; i < N; ++i) {  
    aux[count[a[i].charAt(d)]++] = a[i];  
}  
for (int i = 0; i < N; ++i) {  
    a[i] = aux[i];  
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- *Ascii charset: 256 entries*
 - *Digits charset as an example*
 - *Sort by the right-most digit*
- *Step 2: accumulate the frequencies*
 - *What is count[] mean?*

0: 0	0: 0
1: 5	1: 5
2: 0	2: 5
3: 0	3: 5
4: 1	4: 6
5: 1	5: 7
6: 3	6: 10
7: 0	7: 10
8: 0	8: 10
9: 1	9: 11
10: 2	10: 13



```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - *Digits charset as an example*
 - *Sort by the right-most digit*
- **Step 2: accumulate the frequencies**
 - *What is count[] mean?*

0: 0	Count[0]=0	→	
1: 5			
2: 5			
3: 5			
4: 6	Count[3]=5	→	
5: 7	Count[4]=6	→	
6: 10	Count[5]=7	→	
7: 10			
8: 10	Count[8]=10	→	
9: 11	Count[9]=11	→	
10: 13			

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0	Count[0]=0	→	
1: 5			
2: 5			
3: 5			
4: 6	Count[3]=5	→	
5: 7	Count[4]=6	→	
6: 10	Count[5]=7	→	
7: 10			
8: 10	Count[8]=11	↘	4PGC938
9: 11	Count[9]=11	→	
10: 13			

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0			2IYE230
1: 5	Count[0]=1 →		
2: 5			
3: 5			
4: 6	Count[3]=5 →		
5: 7	Count[4]=6 →		
6: 10	Count[5]=7 →		
7: 10			
8: 10	Count[8]=11 ↘		4PGC938
9: 11	Count[9]=11 →		
10: 13			

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5	Count[0]=2 →	
3: 5		
4: 6	Count[3]=5 →	
5: 7	Count[4]=6 →	
6: 10	Count[5]=7 →	
7: 10		
8: 10	Count[8]=11 ↘	4PGC938
9: 11	Count[9]=11 →	
10: 13		

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- *Ascii charset: 256 entries*
 - *Digits charset as an example*
 - *Sort by the right-most digit*
- *Step 3: copy to output array*

0: 0		2IYE230
1: 5		3CI0720
2: 5	Count[0]=3 →	1ICK750
3: 5		
4: 6	Count[3]=5 →	
5: 7	Count[4]=6 →	
6: 10	Count[5]=7 →	
7: 10		
8: 10	Count[8]=11 ↘	4PGC938
9: 11	Count[9]=11 →	
10: 13		

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
10HV845
4JZY524
1ICK750
3CI0720
10HV845
10HV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=3 →	
4: 6	Count[3]=5 →	
5: 7	Count[4]=6 →	
6: 10		1OHV845
7: 10	Count[5]=8 →	
8: 10	Count[8]=11 ↘	4PGC938
9: 11	Count[9]=11 →	
10: 13		

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=3 →	
4: 6		
5: 7	Count[3]=5 →	
6: 10		4JZY524
7: 10	Count[4]=7 →	1OHV845
8: 10	Count[5]=8 →	
9: 11		
10: 13	Count[8]=11 ↘	4PGC938
	Count[9]=11 →	

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=4 →	1ICK750
4: 6	Count[3]=5 →	
5: 7		4JZY524
6: 10	Count[4]=7 →	1OHV845
7: 10	Count[5]=8 →	
8: 10		
9: 11	Count[8]=11 ↘	4PGC938
10: 13	Count[9]=11 →	

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0			2IYE230
1: 5			3CI0720
2: 5			1ICK750
3: 5	Count[0]=5	↘	1ICK750
4: 6	Count[3]=5	→	3CI0720
5: 7			
6: 10	Count[4]=7	→	4JZY524
7: 10	Count[5]=8	→	1OHV845
8: 10			
9: 11	Count[8]=11	↘	4PGC938
10: 13	Count[9]=11	→	

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0			2IYE230
1: 5			3CI0720
2: 5			1ICK750
3: 5	Count[0]=5	↘	1ICK750
4: 6	Count[3]=5	→	3CI0720
5: 7			
6: 10	Count[4]=7	→	4JZY524
7: 10			1OHV845
8: 10	Count[5]=9	→	1OHV845
9: 11	Count[8]=11	↘	
10: 13	Count[9]=11	→	4PGC938

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=5 ↘	1ICK750
4: 6	Count[3]=5 →	3CI0720
5: 7		
6: 10	Count[4]=7 →	4JZY524
7: 10		1OHV845
8: 10	Count[5]=10 ↘	1OHV845
9: 11	Count[8]=11 ↘	1OHV845
10: 13	Count[9]=11 →	4PGC938

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0			2IYE230
1: 5			3CI0720
2: 5			1ICK750
3: 5	Count[0]=5	↘	1ICK750
4: 6	Count[3]=5	→	3CI0720
5: 7			
6: 10	Count[4]=7	→	4JZY524
7: 10			1OHV845
8: 10	Count[5]=10	↘	1OHV845
9: 11	Count[8]=11	↘	1OHV845
10: 13			4PGC938
	Count[9]=12	→	2RLA629

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=5 ↘	1ICK750
4: 6	Count[3]=5 →	3CI0720
5: 7		
6: 10	Count[4]=7 →	4JZY524
7: 10		1OHV845
8: 10	Count[5]=10 ↘	1OHV845
9: 11	Count[8]=11 ↘	1OHV845
10: 13	Count[9]=13 ↘	4PGC938
		2RLA629
		2RLA629

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=5	1ICK750
4: 6		3CI0720
5: 7	Count[3]=6	3ATW723
6: 10	Count[4]=7	4JZY524
7: 10		1OHV845
8: 10	Count[5]=10	1OHV845
9: 11	Count[8]=11	1OHV845
10: 13	Count[9]=13	4PGC938
		2RLA629
		2RLA629

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Key-indexed counting

- **Ascii charset: 256 entries**
 - **Digits charset as an example**
 - **Sort by the right-most digit**
- **Step 3: copy to output array**

0: 0		2IYE230
1: 5		3CI0720
2: 5		1ICK750
3: 5	Count[0]=5 ↘	1ICK750
4: 6		3CI0720
5: 7	Count[3]=6 →	3ATW723
6: 10	Count[4]=7 →	4JZY524
7: 10		1OHV845
8: 10	Count[5]=10 ↘	1OHV845
9: 11	Count[8]=11 ↘	1OHV845
10: 13	Count[9]=13 ↘	4PGC938
		2RLA629
		2RLA629

```
int [] count = new int[R + 1];
// step1: calculate the histogram of key frequencies
for (int i = 0; i < N; ++i) {
    count[a[i].charAt(d) + 1]++;
}
// step 2: calculate the starting index for each key
for (int r = 0; r < R; ++r) {
    count[r + 1] += count[r];
}
// step 3: copy to output array, preserving order of with equal keys:
for (int i = 0; i < N; ++i) {
    aux[count[a[i].charAt(d)]++] = a[i];
}
for (int i = 0; i < N; ++i) {
    a[i] = aux[i];
}
```

4PGC938
2IYE230
3CI0720
1ICK750
1OHV845
4JZY524
1ICK750
3CI0720
1OHV845
1OHV845
2RLA629
2RLA629
3ATW723

Radix sort

```
public static void sort(String [] a, int W) {  
    int N = a.length;  
    int R = 256;  
    String [] aux = new String[N];  
    for (int d = W - 1; d >= 0; --d) {  
        aux = sorted array a by the dth character  
        a = aux  
    }  
}
```

Radix sort

```
public static void sort(String [] a, int W) {  
    int N = a.length;  
    int R = 256;  
    String [] aux = new String[N];  
  
    for (int d = W - 1; d >= 0; --d) {  
        int [] count = new int[R + 1];  
        for (int i = 0; i < N; ++i) {  
            count[a[i].charAt(d) + 1]++;  
        }  
        for (int r = 0; r < R; ++r) {  
            count[r + 1] += count[r];  
        }  
        for (int i = 0; i < N; ++i) {  
            aux[count[a[i].charAt(d)]++] = a[i];  
        }  
        for (int i = 0; i < N; ++i) {  
            a[i] = aux[i];  
        }  
    }  
}
```

Radix sort analysis

- Runtime:
 - Worst case:
 - $O(nk)$
 - Where k is the length of the strings
- In-place?
 - No
- Stable?
 - Yes
- Limitations?

Questions?