Sorting restricted ranges of numbers

- **If the range is restricted, we can sort using more than comparisons and swaps.**
- **Assume each of the n input elements is an integer in the range $1 \ldots k$.**

Sorting restricted ranges of numbers

- **If the range is restricted, we can sort using more than comparisons and swaps.**
- **Assume each of the n input elements is an integer in the range $1 \ldots k$.**

Idea **For each $A[i]$ compute the number of elements less than or equal to $A[i]$ use that to compute position.**

Sorting restricted ranges of numbers

- **If the range is restricted, we can sort using more than comparisons and swaps.**
- **Assume each of the n input elements is an integer in the range $1 \ldots k$.**

Idea **For each $A[i]$ compute the number of elements less than or equal to $A[i]$, and use that to compute position.**

- **Array $A[1 \ldots n]$ – holds input**
- **Array $C[1 \ldots k]$ – $C[j]$ holds number of elements of A less than or equal to j**

Example:

index	1	2	3	4	5	6	7	8	9
$A:$	2	9	1	8	6	5			

Sorting restricted ranges of numbers

- If the range is restricted, we can sort using more than comparisons and swaps.
- Assume each of the n input elements is an integer in the range $1 \ldots k$.

Idea For each $A[i]$ compute the number of elements less than or equal to $A[i]$ and use that to compute position.

- Array $A[1 \ldots n]$ – holds input
- Array $C[1 \ldots k]$ – $C[j]$ holds number of elements of A less than or equal to j

Example:

index	1	2	3	4	5	6	7	8	9
$A:$	2	9	1	8	6	5			
$C:$	1	2	2	2	3	4	4	5	6

Questions

- How do we compute C?

Counting Sort

```
Counting-Sort(A, B, k)
for i = 0 to k
        C[i] = 0
for j = 1 to length[A]
        C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
        C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = length[A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

Analysis

- **Running Time** $O(n+k)$
- **No Comparisons**
- **Doesn't work on all data**
- **Good when** k **is small**
- **When** $k = O(n)$ **we have run-time** $O(n+k) = O(n)$
- **Examples?**

Stable Sorting

- **We want to sort** $x_1, x_2, ..., x_n$
- **If** $x_i > x_j$ **then put** x_i **after** x_j

Stable Sorting

- **We want to sort** $x_1, x_2, ..., x_n$
- **If** $x_i > x_j$ **then put** x_i **after** x_j
- **But what if** $x_i = x_j$

Stable Sorting

- **We want to sort** $x_1, x_2, ..., x_n$
- **If** $x_i > x_j$ **then put** x_i **after** x_j
- **But what if** $x_i = x_j$

Stable Sorting: **if** $i < j$ **and** $x_i = x_j$ **then put** x_i **before** x_j

Stable Sorting

- **We want to sort** $x_1, x_2, ..., x_n$
- **If** $x_i > x_j$ **then put** x_j **after** x_i
- **But what if** $x_i = x_j$

Stable Sorting: **if** $i < j$ **and** $x_i = x_j$ **then put** x_i **before** x_j

Question: **Is counting sort stable?**

Improving Counting Sort

Question: **Should we use counting sort to sort everyone in this class by initials?**

Improving Counting Sort

Question: **Should we use counting sort to sort everyone in this class by initials?**

- $n = 150$
- $k = 27^2 > 700$
- **Running time is** $150 + 700 = 850$

Improving Counting Sort

Question: **Should we use counting sort to sort everyone in this class by initials?**

- $n = 150$
- $k = 27^2 > 700$
- **Running time is** $150 + 700 = 850$

Improvement: Radix Sort

- **Sort second initial**
- **Then stable sort by first initial.**

Improving Counting Sort

Question: **Should we use counting sort to sort everyone in this class by initials?**

- $n = 150$
- $k = 27^2 > 700$
- **Running time is** $150 + 700 = 850$

Improvement: Radix Sort

- **Sort second initial**
- **Then stable sort by first initial.**

Analysis

- **Sorting a single letter:** $150 + 27 < 200$
- **Total running time:** $2(150 + 27) < 400$

Radix Sort

$Radix - Sort(A, d)$

for $i = 1$ **to** d
    use a stable sort to sort array A **on digit** i

Example

379	**STABLE SORT**	912	**STABLE SORT**	802	**STABLE SORT**	258
912	$\Rightarrow$	802	$\Rightarrow$	803	$\Rightarrow$	259
258		823		804		269
269		803		912		279
823		804		823		379
259		258		258		802
803		269		259		803
279		259		269		804
804		379		379		823
802		279		279		912

Radix Sort Correctness

$Radix - Sort(A, d)$

```
for i = 1 to d
        use a stable sort to sort array A on digit i
```

Loop Invariant: **After the ith iteration of the loop, the elements are sorted by their last i digits.**

Radix Sort Correctness

$Radix - Sort(A, d)$

```
for i = 1 to d
        use a stable sort to sort array A on digit i
```

Loop Invariant: **After the ith iteration of the loop, the elements are sorted by their last i digits.**

Inductive Step:

- **Assume the invariant holds after** $i - 1$ **iterations**
- **Need to prove that it holds after** i **iterations**

Radix Sort Analysis

- n elements

Radix Sort Analysis

- n **elements**
- **All elements have** d **digits**
  - **Initials:** $d = 2$
  - **SSN:** $d = 9$
  - **Dictionary Words:** $d = 30$

Radix Sort Analysis

- n elements
- All elements have d digits
  - Initials: $d = 2$
  - SSN: $d = 9$
  - Dictionary Words: $d = 30$
- Digits are in base b
  - Numbers: $b = 10$
  - Words: $b = 27$
  - UNI (letter/number): $b = 37$

Radix Sort Analysis

- n elements
- All elements have d digits
  - Initials: $d = 2$
  - SSN: $d = 9$
  - Dictionary Words: $d = 30$
- Digits are in base b
  - Numbers: $b = 10$
  - Words: $b = 27$
  - UNI (letter/number): $b = 37$

Radix Sort Running Time: $O(d(n + b))$

Radix Sort Analysis

- n elements
- All elements have d digits
  - Initials: $d = 2$
  - SSN: $d = 9$
  - Dictionary Words: $d = 30$
- Digits are in base b
  - Numbers: $b = 10$
  - Words: $b = 27$
  - UNI (letter/number): $b = 37$

Radix Sort Running Time: $O(d(n + b))$

Counting Sort Running Time: $O(n + k) = O(n + b^d)$

Example

Setup : Sort everyone in columbia by UNI. Say $n = 40,000$

Example

Setup : **Sort everyone in columbia by UNI. Say** $n = 40,000$

Radix Sort:

- $d = 7$
- $b = 37$
- **Running Time:** $d(n + b) = 7(40,000 + 37) \sim 280,000$

Example

Setup : Sort everyone in columbia by UNI. Say $n = 40,000$

Radix Sort:

- $d = 7$
- $b = 37$
- **Running Time:** $d(n + b) = 7(40,000 + 37) \sim 280,000$

Counting Sort:

- **UNI = 7-digit number in base 37.**
- $k = b^d = 37^7 \sim 10^{11}$
- **Running Time:** $n + k = 40,000 + 37^7 \sim 10^{11}$

Example

Setup : Sort everyone in columbia by UNI. Say $n = 40,000$

Radix Sort:

- $d = 7$
- $b = 37$
- **Running Time:** $d(n + b) = 7(40,000 + 37) \sim 280,000$

Counting Sort:

- **UNI = 7-digit number in base 37.**
- $k = b^d = 37^7 \sim 10^{11}$
- **Running Time:** $n + k = 40,000 + 37^7 \sim 10^{11}$

Merge Sort

- **Running Time:** $nlog(n) = 40,000 \cdot \log(40,000) \sim 600,000$