CS 590: Algorithms

Sorting and Order Statistics II:
Counting Sort / Radix Sort / Bucket Sort

Outline

5.1. Counting Sort:

- Counting sort: Non-comparison sorting algorithm.
- Assumption: The input array, A, has n-many integers, and they are from a set {0, 1, …, k}.
- Input: A[1,…, n] where A[j] ∈ {0,…,k} for j = 1,…, n.
- Suppose the array A values, n and k, are given parameters.
- Output: A sorted array B[1,…,n] has an already allocated parameter from an auxiliary storage C[0,…,k].

5.1. Counting Sort:

$A = [2, 5, 3, 0, 2, 3, 0, 3]$

$\{0, 2, 3, 5\}$

$C[A[i]] = C[A[i]] + 1$

$i = 5$

$C[A[5]] = C[A[5]] + 1$

$B = [\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ ]$ empty w/ length = n

$C = [\ \ ]$ empty w/ k

$C = [0, 0, 0, 0, 0]$ (2)

$C = [2, 0, 2, 3, 0, 1]$

$C = [2, 2, 4, 7, 7, 8]$ ← line 7

$C[1] + C[0]$

$C[0] + C[1] + C[2]$

5.1. Counting Sort:

```
Counting-Sort(A,B,n,k)
 1  create new array C[0,…,k]
 2  for (0<=i<=k) do
 3       C[i] = 0
 4  for (1 <= j <= n) do
 5       C[A[j]] = C[A[j]] + 1
 6  for (1 <= i <= k) do
 7         C[i] = C[i] + C[i - 1]
 8  for (n >= j >= 1) do
 9         B[C[A[j]]] = A[j]
10         C[A[j]] = C[A[j]] -1
```

5.1. Counting Sort:

$A = [2, 5, 3, 0, 2, 3, 0, 3]$

$B = [\ .\ .\ -\ -\ -\ ]$ empty w/ length = n

$\{0, 2, 3, 5\}$

$C = [\ ]$ empty w/ k

$C = [0, 0, 0, 0, 0]$ (2 ↓)

$C[A[i]] = C[A[i]] + 1$

$i = 5$

$C[A[5]] = C[A[5]] + 1$

$C = [2, 0, 2, 3, 0, 1]$

$C = [2, 2, 4, 7, 7, 8]$ ← line 7

$C[1] + C[0]$

$C[0] + C[1] + C[2]$

A = [2, 5, 3, 0, 2, 3, 0, 3]
C = [2,0,2,3,0,1] @ L5
C = [2,2,4,7,7,8] @ L7
Remember: A indices start from 1, and C starts from 0

Iteration	j	A[j]	C[A[j]]	B[C[A[j]]	C[A[j]] @ 10	Outcome
1st	8	A[8]=3	C[3]=7	B[7]=3	C[3]=6	C=[2,2,3,6,7,8], B=[_,_,_,_,_,_,3,_]
2nd	7	A[7]=0	C[0]=2	B[2]=0	C[0]=1	C=[1,2,3,6,7,8], B=[_,0,_,_,_,_,3,_]
3rd	6	A[6]=3	C[3]=6	B[3]=3	C[3]=5	C=[1,2,2,5,7,8], B=[_,0,_,_,_,3,3,_]
4th	5	A[5]=2	C[2]=4	B[4]=2	C[2]=3	C=[1,2,1,5,7,8], B=[_,0,_,2,_,3,3,_]
5th	4	A[4]=0	C[0]=1	B[1]=0	C[0]=0	C=[0,2,1,5,7,8], B=[0,0,_,2,_,3,3,_]
6th	3	A[3]=3	C[3]=5	B[5]=3	C[3]=4	C=[0,2,1,4,7,8], B=[0,0,_,2,3,3,3,_]
7th	2	A[2]=5	C[5]=8	B[8]=5	C[5]=7	C=[0,2,1,4,7,7], B=[0,0,_,2,3,3,3,5]
8th	1	A[1]=2	C[2]=3	B[3]=2	C[2]=2	C=[0,2,0,4,7,7], B=[0,0,2,2,3,3,3,5]

5.1. Counting Sort:

- The counting sort is stable.
  - keys with the same value appear in the same order in the output as in the input.
- The last loop in the algorithm ensures this property.
- Insertion sort and merge sort are stable sorting algorithms.

5.1. Counting Sort:

Analysis:

- Running time $T(\mathrm{n}) = \Theta(n + k)$.
- If k = n, the running time becomes $\Theta(n)$.

Is it practical?

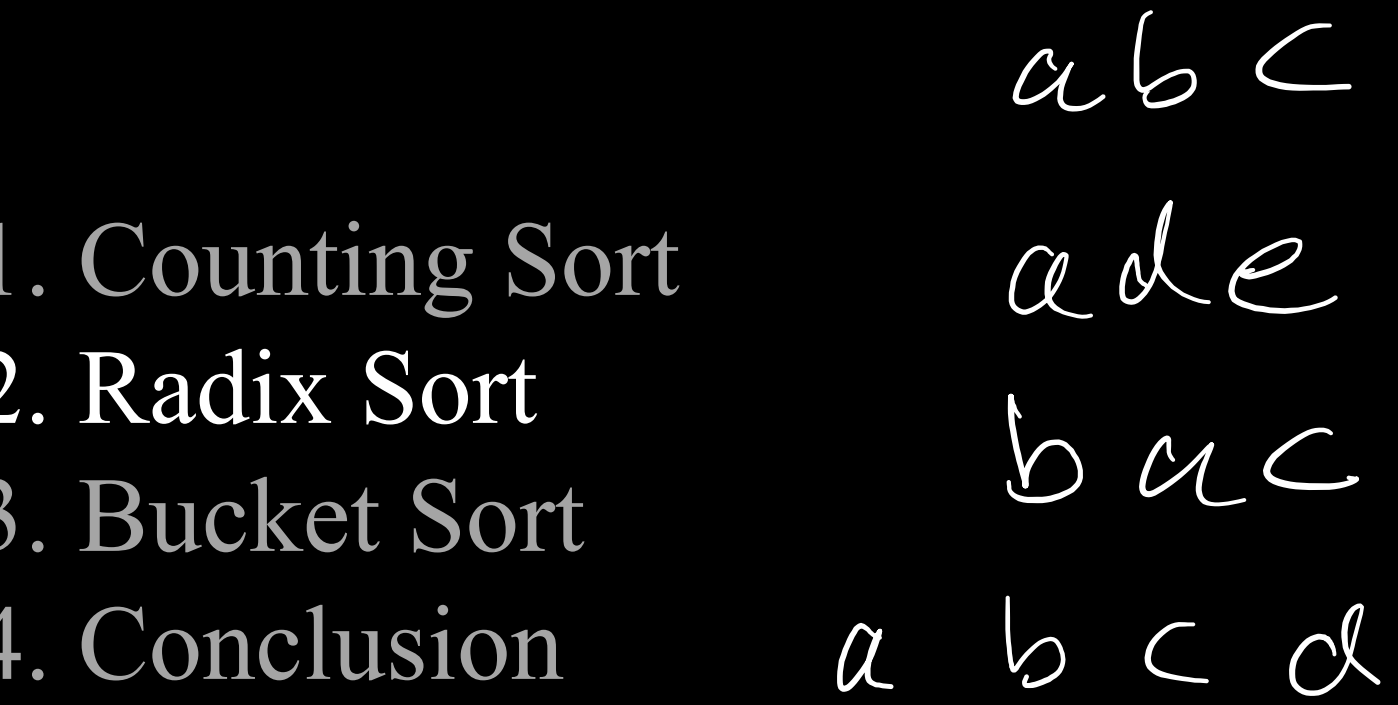
- Not a good idea to use it to sort 32-bit or 16-bit values.
- Probably a good idea for an 8-bit or 4-bit value.
- It is strongly depends on the number of values n.

5.1. Counting Sort:

Memory consumption can be a problem.

- The auxiliary storage C necessary for goes from 0 to k.
- For the 32-bit integers, we need 16 GB of auxiliary storage.
- We need a 32-bit counter for each integer from 0 to k, $2^{32} - 1$.
- $\Rightarrow$ We will use counting sort within radix sort.

5.2. Radix Sort

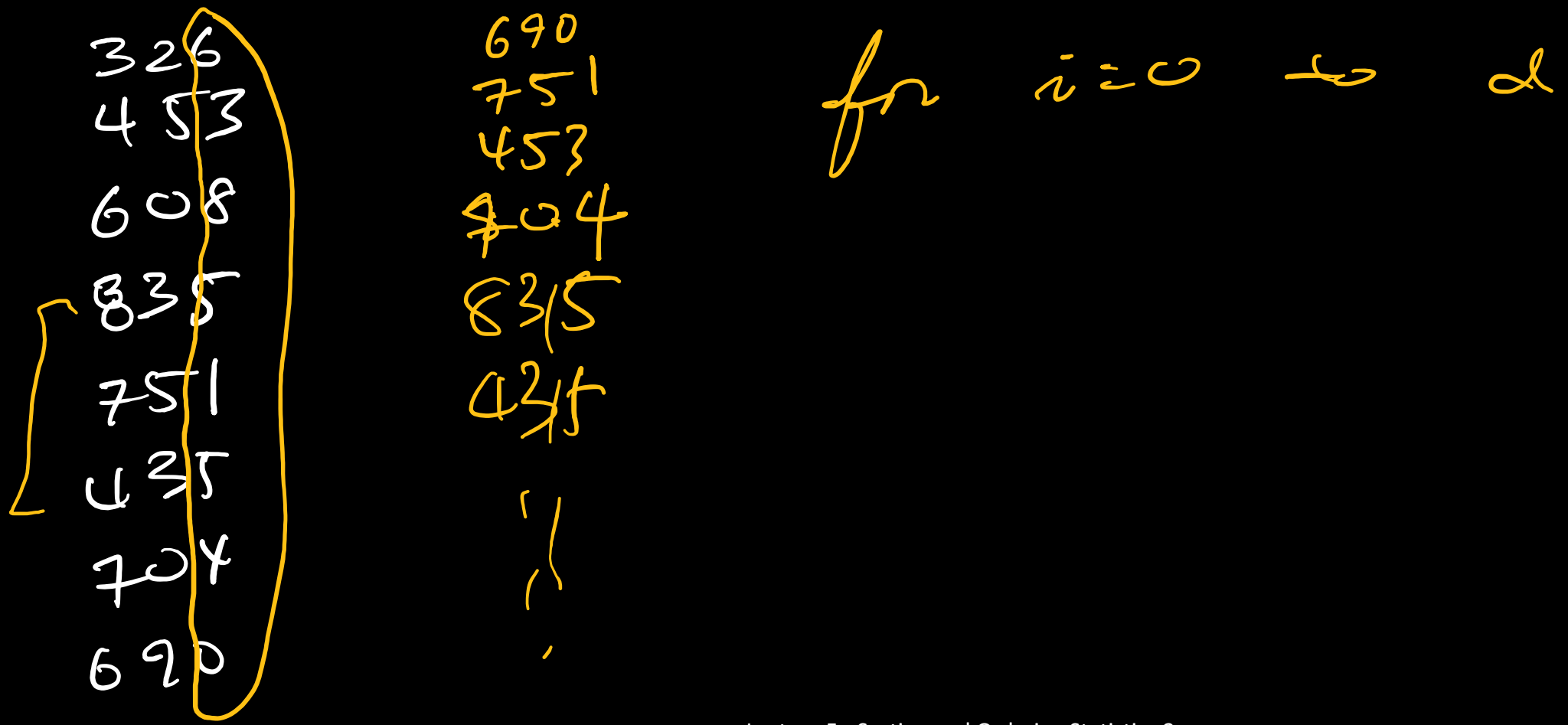


5.2. Radix Sort

- It goes back to IBM and the census in early 1900.
- Assume the input is integers with multiple digits, e.g., 112, 241, etc.
- A random integer with d digits is generated. There are 9*10^(d-1) different ways.
- If an input is a string, then a n-length string will have 26^n different ways to make.

5.2. Radix Sort

Consider an array A=[326, 453, 608, 835, 751, 435, 704, 690].



5.2. Radix Sort

```
RADIX-SORT(A, d)
1   for (1 <= j <= d) do
2       //* call stable sort algorithm
        sort on digit j
```

$d(T(n))$

↑ sorting algorith

5.2. Radix Sort

Correctness:

- We use induction on the number of passes (loop variable j).
- Suppose that the digits 1,…, j-1 are sorted.
- Then, a stable sorting algorithm on digit j leaves the digits 1,…, j-1 sorted.
  - If two digits on j are different, ordering by position j is correct.
    - Positions 1,…, j-1 are irrelevant.
  - If two digits in position j are equal, then the numbers are already in the right order (by inductive hypothesis).
    - The stable sort on digit j leaves them in the right order.

5.2. Radix Sort

Analysis:

- Assume that the counting sort is used as the intermediate sort.
- Running time $\Theta(n + k)$ per pass (digits in range $0, \ldots, k$).
- The for-loop passes d many times.
- The total running time is $T(n) = \Theta\big(d(n + k)\big)$.
  - If $k = n$, then $T(n) = \Theta(dn)$.

5.2. Radix Sort

Break values into digits:

- Suppose we have n words with b bits per word.
- We break into r-bit digits, then the digits are $d = \left\lceil \frac{b}{r} \right\rceil$.
- We use counting sort with $k = 2^r - 1$.
- Example: 32-bit words, 8-bit digits.
  - $b = 32, r = 8, d = 4, k = 2^8 - 1 = 255$.
- Running time: $\Theta\left(\frac{b}{r}(n + 2^r)\right)$ for Radix sort.

5.2. Radix Sort

How do we choose r?

- We have to balance $\frac{b}{r}$ and $n + 2^r$.
- Choose $r \approx \lg n \Rightarrow \Theta\left(\frac{b}{\lg n}(n+n)\right) = \Theta\left(\frac{bn}{\lg n}\right)$ running time.
- Choose $r < \lg n$, then $\frac{b}{r} > \frac{b}{\lg n}$ and the term $n + 2^r$ does not improve.
- Choose $r > \lg n$, then the term $n + 2^r$ increases. For $r = 2\lg n$ we get $2^r = 2^{2\lg n} = \left(2^{\lg n}\right)^2 = n^2$.

Example: If we sort 2^{16} numbers of size 32-bit, we use $r = \lg 2^{16} = 16$ bits.

We perform $\left\lceil \frac{b}{r} \right\rceil = 2$ passes.

5.2. Radix Sort

How does radix sort compare to merge sort and quicksort?

- For 1 million ($\approx 2^{20}$) 32 bit integers.
- Radix sort performs $\left\lceil \frac{32}{20} \right\rceil = 2$ passes, whereas merge or quicksort perform $\lg n = 20$ passes.
- One radix sort is really 2 passes ⇒ one to take and one to move data.

How radix sort violates comparison sort rules?

- Using counting sort allows us to gain information without directly comparing 2 keys.
- Use the keys as array indices.

5.3. Bucket Sort

5.1. Counting Sort
5.2. Radix Sort
5.3. Bucket Sort
5.4. Conclusion

Bucket Sort

We assume the input is generated randomly and the elements are distributed uniformly over $[0,1)$.

Idea:

- We divide the interval $[0,1)$ into n equal-sized buckets.
- We distribute the n input values into the buckets.
- We sort each bucket afterward.
- And then, we go through the buckets in order, listing the elements in each one.

Bucket Sort

```
RADIX-SORT(A, d, n)
 1  create new array B[0,..,n-1] of lists
 2  for (0 <= i <= n-1) do
 3      let B[i] be an empty list
 4  for (1 <= i <= n) do
 5      put (insert) A[i] into bucket
 6  B[floor(nA[i])]
 7  for (0 <= i <= n-1) do
 8      sort list B[i] with insertion sort
 9  concatenate list B[0],…,B[n-1] together in order
10  return concatenated list
```

Bucket Sort

Correctness:

- We consider $A[i], A[j]$ (assume $A[i] \leq A[j]$).
- $\lfloor n \cdot A[i] \rfloor \leq \lfloor n \cdot A[j] \rfloor$ follows.
- $A[i]$ is placed in same or in bucket with lower index than $A[j]$.
  - same bucket ⇒ insertion sort fixes order.
  - earlier bucket ⇒ concatenation of list ensures order.

5.3. Bucket Sort

Example:

A[1…10]=[0.89, 0.13, 0.45, 0.2, 0.54, 0.53, 0.7, 0.85, 0.51, 0.49]

- Initialize B[0…9]=0,…,0
- Put A[i] into the 10 buckets:

B[0]	B[1]	B[2]	B[4]	B[5]
0	0.13	0.2	0.45, 0.49	0.54, 0.53, 0.51
B[6]	B[7]	B[8]	B[9]	
0	0.7	0.89, 0.85	0	

5.3. Bucket Sort

- Sort each of the 10 buckets:

B[0]	B[1]	B[2]	B[4]	B[5]
0	0.13	0.2	0.45, 0.49	0.51, 0.53,0.54
B[6]	B[7]	B[8]	B[9]	
0	0.7	0.85, 0.89		

- Concatenate all of the buckets: B[0…9]=0.13, 0.2, 0.45, 0.51, 0.53, 0.54, 0.7, 0.85, 0.89

5.3. Bucket Sort

Analysis:

- The algorithm relies on the fact that no bucket is getting too many values (uniformly distributed).
- The total running time (except the insertion sort) is $\Theta(n)$.
- Intuition: Each bucket gets a constant number of elements. Sorting then takes constant time for each bucket.
- Using our intuition, we then get $O(n)$ sorting time for all buckets.
- On average, we have 1 element per bucket. ⇒ we expect each bucket to have a few elements.

⇒ We need to do a careful analysis.

$T(n) = n \cdot$ Sorting al. in $B[i]$ $< O(c^2)$

Full

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} \text{insertion sort worst case}$$

↑ $T(n)$ for Bucket creation

Considering the worst case of insertion sort w/ n_i:

let n_i be the number of elements in $B[i]$

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

E : expected value for Gaussian, it is the mean

$$E(T(n)) = E(\Theta(n)) + \sum_{i=0}^{n-1} O(E(n_i^2))$$

let X_{ij} in $\{A[j] \text{ in } B[i]\}$

The probability of $A[j]$ in $B[i]$ is $1/n$ +

$$n_i = \sum_{j=1}^{n} X_{ij} = X_{i1} + X_{i2} + \cdots + X_{in}$$

$E(n_i^2)$ is the combination of

$$n_i^2 = 2\sum_{j=1}^{n} X_{ij}^2 + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} X_{ij} X_{ik}$$

↑ square

regroup: for $i \neq j$

$$E(n_i^2) = \sum E(X_{ij}^2) + 2\sum\sum E(X_{ij} X_{ik})$$

for $E(\sum X_{ij}^2)$: $1/n$

for $E(\sum X_{ij} X_{ik}) = E(X_{ij})E(X_{ik}) = 1/n \cdot 1/n$

$$E(n_i^2) = \underbrace{\sum 1/n}_{= n \cdot 1/n = 1} + \underbrace{\sum\sum 1/n^2}_{n(n-1)/n^2 = \text{Const} + 1/n}$$

$$E(T(n)) = \Theta(n) + \sum_{i=0}^{n-1} [\text{const} + 1/n]$$

$$= \Theta(n) + n \cdot \text{const} + 1 \quad \therefore \Theta(n)$$