

CS5101: Design and Analysis of Algorithms

Sorting in Linear Time



Arijit Mondal

Dept of CSE

`arijit@iitp.ac.in`

`https://www.iitp.ac.in/~arijit/`

Sorting

- Lower bounds for comparison based sorting in $\Omega(n \lg n)$
- Sorting can be done in linear time $O(n)$ when we have some idea about the range of the numbers
- Knowledge of distribution of numbers can be exploited to improve the efficiency

Linear Time Sorting

- Consider an array $A = [4, 2, 2, 8, 3, 3, 1]$
- The range of the numbers is between 1 and 8
- Let us count the number of numbers for each of the numbers

Value	1	2	3	4	5	6	7	8
$C[x]$	1	2	2	1	0	0	0	1

- Sorted sequence can be generated by printing '1' one time, '2' two times, '3' two times, and so on
- However, this is not a good idea**

Counting sort

- Consider an array $A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$, where the suffix denotes some additional information associated with the key
- The range of the keys is between 1 and 8
- Let us count the number of numbers for each of the numbers (same as previous)

Value	1	2	3	4	5	6	7	8
$C[x]$	1	2	2	1	0	0	0	1

Counting sort

- Consider an array $A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$, where the suffix denotes some additional information associated with the key
- The range of the keys is between 1 and 8
- Let us count the number of numbers for each of the numbers (same as previous)

Value	1	2	3	4	5	6	7	8
$C[x]$	1	2	2	1	0	0	0	1

- Cumulative count – number of elements less or equals to i

Value	1	2	3	4	5	6	7	8
$C[x]$	1	3	5	6	6	6	6	7

Counting sort

- Consider an array $A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$, where the suffix denotes some additional information associated with the key
- The range of the keys is between 1 and 8
- Let us count the number of numbers for each of the numbers (same as previous)

Value	1	2	3	4	5	6	7	8
$C[x]$	1	2	2	1	0	0	0	1

- Cumulative count – number of elements less or equals to i

Value	1	2	3	4	5	6	7	8
$C[x]$	1	3	5	6	6	6	6	7

- Now take elements from A put it in new array B . We will start from the end of A

Couting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	1	3	5	6	6	6	6	7

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B							

Couting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	3	5	6	6	6	6	7

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H						

Counting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	3	4	6	6	6	6	7

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H				3_G		

Couting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	3	3	6	6	6	6	7

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H			3_D	3_G		

Counting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	3	3	6	6	6	6	6

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H			3_D	3_G		8_B

Counting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	2	3	6	6	6	6	6

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H		2_E	3_D	3_G		8_B

Couting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	1	3	6	6	6	6	6

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H	2_C	2_E	3_D	3_G		8_B

Couting sort

$$A = [4_A, 2_C, 2_E, 8_B, 3_D, 3_G, 1_H]$$

Index	1	2	3	4	5	6	7	8
Cumulative value, $C[x]$	0	1	3	5	6	6	6	6

Index	1	2	3	4	5	6	7
A	4_A	2_C	2_E	8_B	3_D	3_G	1_H
B	1_H	2_C	2_E	3_D	3_G	4_A	8_B

Counting sort

- Counting-Sort(A, n, K)
 1. $B[1 : n], C[1 : k]$ – two arrays
 2. $C[i] = 0 \quad \forall i = 1, \dots, k$
 3. for $j=1$ to n
 4. $C[A[j]] = C[A[j]] + 1$
 5. for $i=2$ to k
 6. $C[i] = C[i] + C[i - 1]$
 7. for $j=n$ to 1
 8. $B[C[A[j]]] = A[j]$
 9. $C[A[j]] = C[A[j]] - 1$
 10. return B

Counting sort

- Counting-Sort(A, n, K)
 1. $B[1 : n], C[1 : k]$ – two arrays
 2. $C[i] = 0 \quad \forall i = 1, \dots, k$
 3. for $j=1$ to n
 4. $C[A[j]] = C[A[j]] + 1$
 5. for $i=2$ to k
 6. $C[i] = C[i] + C[i - 1]$
 7. for $j=n$ to 1
 8. $B[C[A[j]]] = A[j]$
 9. $C[A[j]] = C[A[j]] - 1$
 10. return B

- Complexity – $\Theta(n + k)$

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamond < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamond < \dots < K\diamond < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamondsuit < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamondsuit < \dots < K\diamondsuit < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$
- Steps:
 1. Create 13 piles, one for each value (face up)

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamondsuit < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamondsuit < \dots < K\diamondsuit < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$
- Steps:
 1. Create 13 piles, one for each value (face up)
 2. Create single pile, aces at the bottom, then 2s, 3s (face up)

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamondsuit < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamondsuit < \dots < K\diamondsuit < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$
- Steps:
 1. Create 13 piles, one for each value (face up)
 2. Create single pile, aces at the bottom, then 2s, 3s (face up)
 3. Turn the deck face down

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamondsuit < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamondsuit < \dots < K\diamondsuit < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$
- Steps:
 1. Create 13 piles, one for each value (face up)
 2. Create single pile, aces at the bottom, then 2s, 3s (face up)
 3. Turn the deck face down
 4. Deal again, this time with 4 piles, one for each suit

Radix sort

- Suppose a random deck of 52 cards is given. Goal is to order them as per following
 - Rule 1: $A < 2 < 3 < \dots < 10 < J < Q < K$
 - Rule 2: $\clubsuit < \diamondsuit < \heartsuit < \spadesuit$
 - Ordering: $A\clubsuit < \dots < K\clubsuit < A\diamondsuit < \dots < K\diamondsuit < A\heartsuit < \dots < K\heartsuit < A\spadesuit < \dots < K\spadesuit$
- Steps:
 1. Create 13 piles, one for each value (face up)
 2. Create single pile, aces at the bottom, then 2s, 3s (face up)
 3. Turn the deck face down
 4. Deal again, this time with 4 piles, one for each suit
 5. Create a single pile, clubs at the bottom, then diamonds, hearts, spades

Radix sort: Example

479

720

832

329

678

525

347

Radix sort: Example

479

720

832

329

678

525

347

Radix sort: Example

479		720
720		832
832		525
329	→	347
678		678
525		479
347		329

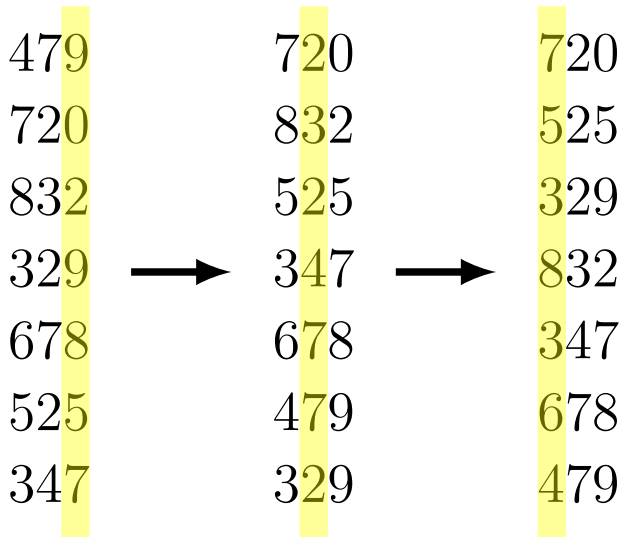
Radix sort: Example

479	→	720
720		832
832		525
329		347
678		678
525		479
347		329

Radix sort: Example

479	→	720	→	720
720		832		525
832		525		329
329		347		832
678		678		347
525		479		678
347		329		479

Radix sort: Example



Radix sort: Example

479	→	720	→	720	→	329
720		832		525		347
832		525		329		479
329		347		832		525
678		678		347		678
525		479		678		720
347		329		479		832

Radix sort

- Radix-Sort(A, n, d)
 1. for $i=1$ to d
 2. use a stable sort to sort $A[1 : n]$ on digit i

Radix sort

- Radix-Sort(A, n, d)
 1. for $i=1$ to d
 2. use a stable sort to sort $A[1 : n]$ on digit i
- Complexity – $\Theta(d(n + k))$, k is the range

Radix sort

- Radix-Sort(A, n, d)
 1. for $i=1$ to d
 2. use a stable sort to sort $A[1 : n]$ on digit i
- Complexity – $\Theta(d(n + k))$, k is the range
- Given n b -bit numbers and any positive integer $r \leq b$, Radix sort correctly sorts these numbers in $\Theta((b/r)(n + 2^r))$ time if the stable sort it uses takes $\Theta(n \log k)$ time for inputs in the range 0 to k

Bucket sort

- Consider $A = [0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68]$
- Assume the number lies between 0 and 1, and uniformly distributed in this range. How to sort efficiently?

Bucket sort

- Consider $A = [0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68]$
- Assume the number lies between 0 and 1, and uniformly distributed in this range. How to sort efficiently?
- Idea:
 - Create n equal size bucket
 - Put $A[i]$ in bucket $\lfloor n \times A[i] \rfloor$
 - Sort each bucket using insertion sort
 - Concatenate all buckets

Bucket sort

- Consider $A = [0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68]$
- Assume the number lies between 0 and 1, and uniformly distributed in this range. How to sort efficiently?
- Idea:
 - Create n equal size bucket
 - Put $A[i]$ in bucket $\lfloor n \times A[i] \rfloor$
 - Sort each bucket using insertion sort
 - Concatenate all buckets
- Complexity: worst case – $O(n^2)$, average case – $O(n)$

Thank you!