Linear Sorting

Dr Timothy Kimber

March 2015

Recalling Comparison Sorts

The running time of these comparison sort algorithms

- Mergesort
- Heapsort
- Quicksort (expected)

are all $O(N \log N)$.

- Not possible for a comparison sort algorithm to do better

However, there are sorting methods that achieve $O(N)$ performance.

Counting Sort

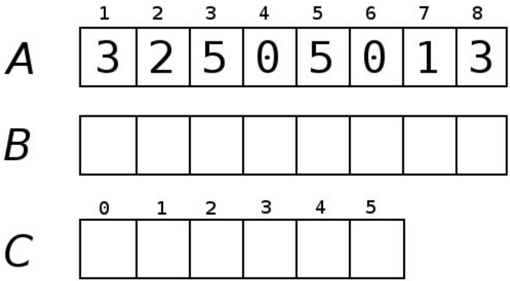
The Counting Sort algorithm sorts integers from a known range

- The key operation is to count the occurrences of all values

Counting Sort(Input: $A = [A_1, \ldots, A_N]$, k)

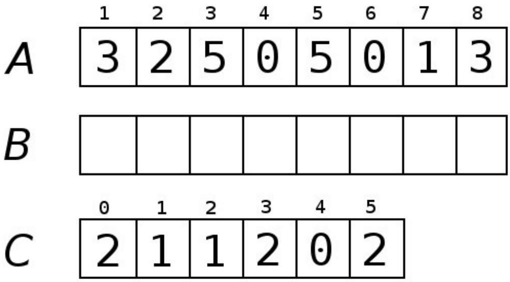
- For $i = 0$ to k
  - $C[i] = 0$ `<-- one entry per value in the range`
- For $j = 1$ to N
  - $C[A[j]] = C[A[j]] + 1$ `<-- count how many A[j] there are`
- For $i = 0$ to k
  - $C[i] = C[i] + C[i-1]$ `<-- how many less than or equal to i`
- For $j = N$ to 1
  - $B[C[A[j]]] = A[j]$
  - $C[A[j]] = C[A[j]] - 1$
- Return B

Counting Sort



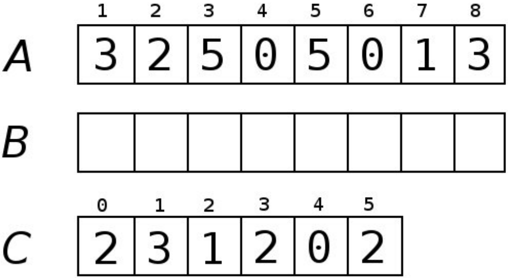
- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

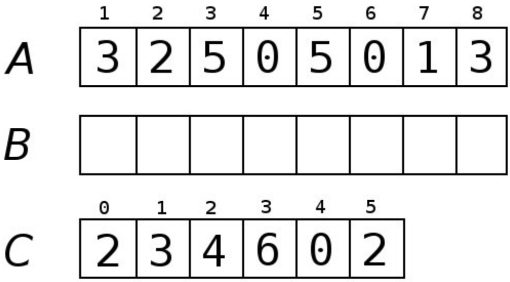
Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B								

	0	1	2	3	4	5
C	2	3	4	2	0	2

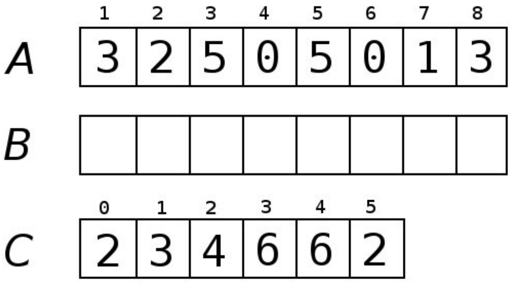
- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B								

	0	1	2	3	4	5
C	2	3	4	6	6	8

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

A

1	2	3	4	5	6	7	8
3	2	5	0	5	0	1	3

B

C

0	1	2	3	4	5
2	3	4	6	6	8

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

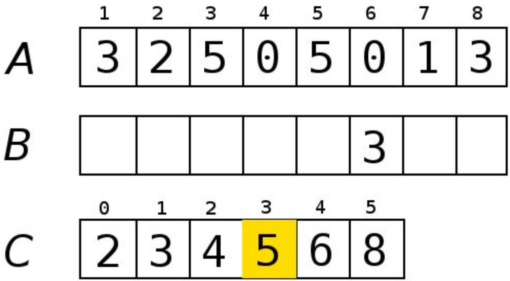
Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B						3		

	0	1	2	3	4	5
C	2	3	4	6	6	8

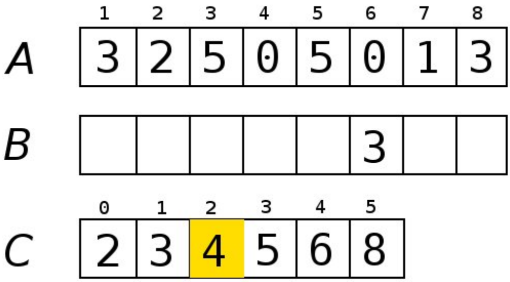
- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B				2		3		

	0	1	2	3	4	5
C	2	3	4	5	6	8

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

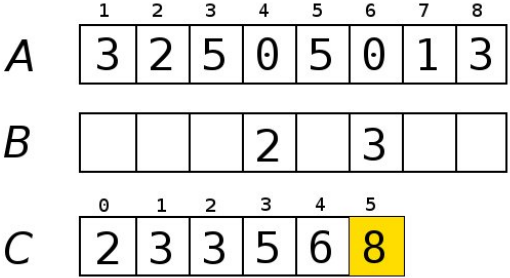
Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B				2		3		

	0	1	2	3	4	5
C	2	3	3	5	6	8

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B				2		3		5

	0	1	2	3	4	5
C	2	3	3	5	6	8

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B				2		3		5

	0	1	2	3	4	5
C	2	3	3	5	6	7

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

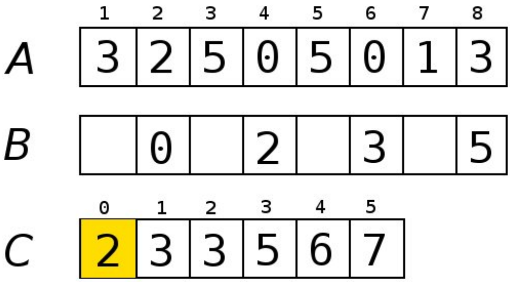
	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3

B				2		3		5

	0	1	2	3	4	5
C	2	3	3	5	6	7

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

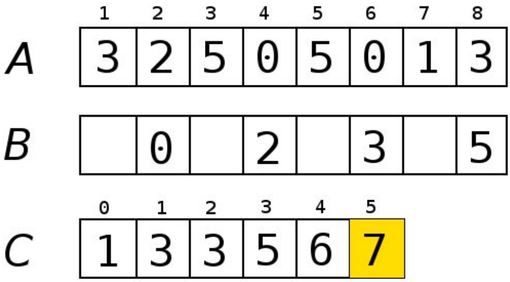
Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B		0		2		3		5

	0	1	2	3	4	5
C	1	3	3	5	6	7

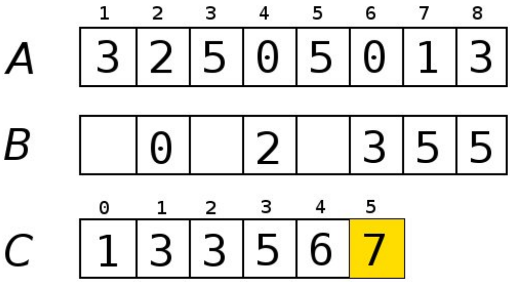
- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B		0		2		3	5	5

	0	1	2	3	4	5
C	1	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3

B		0		2		3	5	5

	0	1	2	3	4	5
C	1	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0		2		3	5	5

	0	1	2	3	4	5
C	1	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0		2		3	5	5

	0	1	2	3	4	5
C	0	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0		2		3	5	5

	0	1	2	3	4	5
C	0	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0	1	2		3	5	5

	0	1	2	3	4	5
C	0	3	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0	1	2		3	5	5

	0	1	2	3	4	5
C	0	2	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

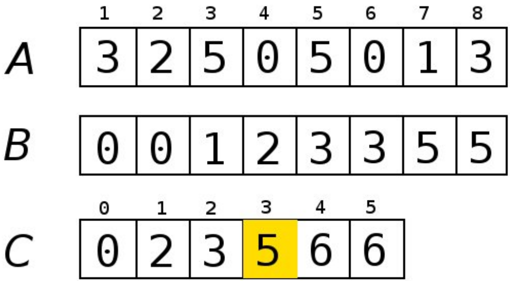
Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0	1	2		3	5	5

	0	1	2	3	4	5
C	0	2	3	5	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort



- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Counting Sort

	1	2	3	4	5	6	7	8
A	3	2	5	0	5	0	1	3
B	0	0	1	2	3	3	5	5

	0	1	2	3	4	5
C	0	2	3	4	6	6

- Counts of each value are saved into C
- Next the counts are accumulated
- Now $C[i]$ holds number of values $\leq i$
- Finally copy contents of A to correct positions in B using C

Performance

Counting Sort runs in $\Theta(N + k)$ time.

- Assuming $k = O(N)$ this bound becomes $\Theta(N)$

Counting Sort is also stable

- 'Different' 3s stay in the same order
- The animation gets this wrong! (Sorry)
- Copying should iterate from $A[N]$ to $A[1]$

Radix Sort

Radix Sort is used to sort a set of d-digit values

$$\begin{array}{ccc} 535 & & 089 \\ 158 & & 134 \\ 189 & & 158 \\ 134 & \rightarrow & 189 \\ 840 & & 535 \\ 558 & & 558 \\ 089 & & 840 \end{array}$$

- It makes d passes through the data
- Each pass sorts on the ith digit only

Radix Sort

Radix Sort is used to sort a set of d-digit values

53**5**
15**8**
18**9**
13**4**
84**0**
55**8**
08**9**

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

Radix Sort is used to sort a set of d-digit values

84**0**
13**4**
53**5**
15**8**
55**8**
18**9**
08**9**

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

Radix Sort is used to sort a set of *d*-digit values

8**4**0
1**3**4
5**3**5
1**5**8
5**5**8
1**8**9
0**8**9

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

Radix Sort is used to sort a set of d-digit values

1**3**4
5**3**5
8**4**0
1**5**8
5**5**8
1**8**9
0**8**9

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

Radix Sort is used to sort a set of *d*-digit values

134
535
840
158
558
189
089

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

Radix Sort is used to sort a set of d-digit values

089
134
158
189
535
558
840

- Counter-intuitively, the first sort is on the least significant digit
- The sort used must be stable

Radix Sort

The algorithm is simple to state

Radix Sort(Input: $A = [A_1, \ldots, A_N]$, d)

- For $i = 0$ to d
  - Use a stable sort to sort A on digit i

- Counting Sort can implement the stable sort efficiently
- e.g. For decimal numbers there are 10 values to sort on

Performance

Assuming we have N numbers

- Each comprising b bits
- Where r bits make one digit

Radix Sort runs in $\Theta((b/r)(N + 2^r))$ time if the stable sort takes $\Theta(N + k)$ time to sort values in the range $0 \ldots k$.

- Each number has b/r digits
- Each digit is in the range $0 \ldots 2^r - 1$
- Counting sort runs in $\Theta(N + k) = \Theta(N + 2^r)$ time
- There are $d = b/r$ passes
- The total time is $\Theta((b/r)(N + 2^r))$

If $b = O(N)$ then r can be chosen as $\log_2 N$ and the running time is $\Theta(N)$. In practice, constant factors often mean that Quicksort is faster.