Linear Sorting

March 2018

Recalling Comparison Sorts

The running time of these comparison sort algorithms

- Mer
- Hea
- Qui

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

- Not possible for a comparison sort algorithm to d

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
  -
- For
  - $C[A[j]] = C[A[j]] + 1$ `<-- cou`
- For $i = 1$ to k
  - $C[i] = C[i] + C[i-1]$ `<-- how many 1`
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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 Time

Counting sort makes two passes through the input and two passes through the count table C

- So the time taken is ...

Countin

- For
  -
- For $j = 1$ to N
  - $C[A[j]] = C[A[j]] + 1$ `<-- cou`
- For $i = 1$ to k
  - $C[i] = C[i] + C[i-1]$ `<-- how many l`
- For $j = N$ to 1
  - $B[C[A[j]]] = A[j]$
  - $C[A[j]] = C[A[j]] - 1$
- Return B

Properties

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

Question

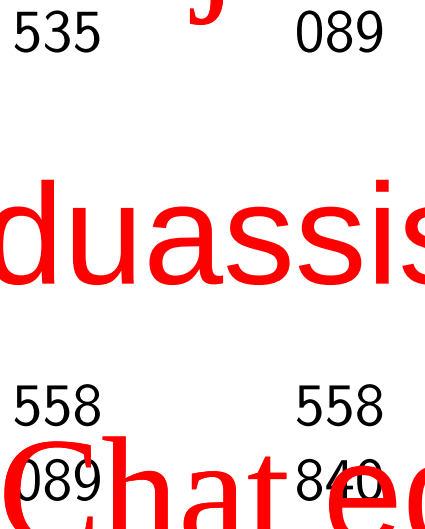
Under wh

Countin

- 'Different' 3s stay in the same order
- Can be important when the values are linked to oth
- This property is used by the next algorithm

Radix Sort

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



- 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

535
158

558
089

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

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

840
134

189
089

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

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

8**4**0
1**3**4

1**8**9
0**8**9

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

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

134
535

189
089

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

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

134
535

189
089

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

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

089
134

558
840

- Counter-intuitively, the first sort is on the least significant digit
- It allows counting sort to be used per digit, over a much smaller range
- e.g. For decimal numbers there are 10 values to sort on

Radix Sort

The algorithm is simple to state

Radix Sor

- For
  - Use a stable sort to sort A on digit i

- Counting Sort can implement the stable sort effi

The Radix

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

- For $i = 0$ to d
  -

Discussi

You are sorting N numbers with Radix sort. You c se the numbers will be represented in within the sort procedur

- What base would you choose?
- Why?

The Radix

Assuming we have N numbers

- Exp
- Eac

Radix sort t

- Base B has values in the range 0 to (
- So, there are B distinct values to count

A base that is $O(N)$, e.g. base N, will limit th
to some smaller base, while not dominating the time for each pass.

Binary

Binary representation allows you to pick any power of 2 as a base very cheaply. Assuming we have N numbers

- Eac
- Spli

Radix Sor

$\Theta(N + k)$ time to sort values in the range 0

- Each number has b/r digits
- Choose $r \sim \log_2(N)$ gives $\sim N$ values

Under the assumption that $b = O(\log_2 N)$ the running time of Radix Sort is $\Theta(N)$. In practice, constant factors may mean that Quicksort is faster.