Analysis of Algorithms

Sorting in linear time

Andres Mendez-Vazquez

July 2, 2018

Outline

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{l} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \quad (1)$$

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{c} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \tag{1}$$

What if we do something different?

How to avoid to compare?

1. You can try to use the relative position of the building blocks of the numbers using certain numeric system!!!
2. You can use the idea of histograms!!!

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{l} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \qquad (1)$$

What if we do something different?

How to avoid to compare?

Maybe, we can do the following

1. You can try to know the position of the numbers in someway !!!
2. You can try to use the relative position of the building blocks of the numbers using certain numeric system !!!
3. You can use the idea of histograms !!!

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{l} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \tag{1}$$

What if we do something different?

How to avoid to compare?

Maybe, we can do the following

1. You can try to know the position of the numbers in someway !!!
2. You can try to use the relative position of the building blocks of the numbers using certain numeric system !!!
3. You can use the idea of histograms !!!

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{l} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \tag{1}$$

What if we do something different?

How to avoid to compare?

Maybe, we can do the following

1. You can try to know the position of the numbers in someway !!!
2. You can try to use the relative position of the building blocks of the numbers using certain numeric system !!!
3. You can use the idea of histograms !!!

About Sorting

Until now we had the following constraint

Given two numbers x and y, we compare to know if

$$\left.\begin{array}{l} x < y \\ x > y \\ x = y \end{array}\right\} \text{The Trichotomy Law} \tag{1}$$

What if we do something different?

How to avoid to compare?

Maybe, we can do the following

1. You can try to know the position of the numbers in someway !!!
2. You can try to use the relative position of the building blocks of the numbers using certain numeric system !!!
3. You can use the idea of histograms !!!

Outline

Counting to find the Position

Imagine the Following

4	9	2	0	2	0	1	4	6	2	9	10	11	6	6	7

Put the Blocks with Equal Elements Together in Clusters

Counting to find the Position

Imagine the Following

4	9	2	0	2	0	1	4	6	2	9	10	11	6	6	7

Put the Blocks with Equal Elements Together in Clusters

		2		6				
0		2	4	6		9		
0	1	2	4	6	7	9	10	11

Accumulate to Know the Clusters of Numbers

Accumulate Them

								11
							10	10
						9	9	9
						9	9	9
					7	7	7	7
				6	6	6	6	6
				6	6	6	6	6
				6	6	6	6	6
			4	4	4	4	4	4
			4	4	4	4	4	4
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

We need to store this information efficiently!!!

Accumulate to Know the Clusters of Numbers

Accumulate Them

								11
							10	10
						9	9	9
						9	9	9
					7	7	7	7
				6	6	6	6	6
				6	6	6	6	6
				6	6	6	6	6
			4	4	4	4	4	4
			4	4	4	4	4	4
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

We need to store this information efficiently!!!

2	1	3	0	2	0	3	1	0	2	1	1
0	1	2	3	4	5	6	7	8	9	10	11

Accumulate to Know the Clusters of Numbers

Accumulate Them

								11
							10	10
						9	9	9
						9	9	9
					7	7	7	7
				6	6	6	6	6
				6	6	6	6	6
				6	6	6	6	6
			4	4	4	4	4	4
			4	4	4	4	4	4
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
		2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

We need to store this information efficiently!!!

2	1	3	0	2	0	3	1	0	2	1	1
0	1	2	3	4	5	6	7	8	9	10	11

Then, we have that

Accumulated Array

2	3	6	6	8	8	11	12	12	14	15	16
0	1	2	3	4	5	6	7	8	9	10	11

We have the basic process there

We only need to be smart about it!!!

Then, we have that

Accumulated Array

2	3	6	6	8	8	11	12	12	14	15	16
0	1	2	3	4	5	6	7	8	9	10	11

We have the basic process there

We only need to be smart about it!!!

Therefore

We have the following limitations

- This technique is good for sorting integers in a narrow range.
- It assumes the input numbers (keys) are in the range $0, ..., k$.

Therefore

We have the following limitations

- This technique is good for sorting integers in a narrow range.
- It assumes the input numbers (keys) are in the range $0, ..., k$.

We need an auxiliary array

$C[0, ..., k]$ to hold the number of items less than i for $0 \leq i \leq k$.

Therefore

We have the following limitations

- This technique is good for sorting integers in a narrow range.
- It assumes the input numbers (keys) are in the range $0, ..., k$.

We need an auxiliary array

$C[0, ..., k]$ to hold the number of items less than i for $0 \leq i \leq k$.

Stable Property

Counting sort is stable; it keeps records in their original order.

Therefore

We have the following limitations

- This technique is good for sorting integers in a narrow range.
- It assumes the input numbers (keys) are in the range $0, ..., k$.

We need an auxiliary array

$C[0, ..., k]$ to hold the number of items less than i for $0 \leq i \leq k$.

Stable Property

Counting sort is stable; it keeps records in their original order.

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C[0..k]$ be a new array
2. **for** $i = 0$ **to** k
3. $\quad C[i] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\quad C[A[j]] = C[A[j]] + 1$
6. $//C[i]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\quad C[i] = C[i] + C[i-1]$
9. $//\ C[i]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\quad B[C[A[j]]] = A[j]$
12. $\quad C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C[0..k]$ be a new array
2. **for** $i = 0$ **to** k
3. $\quad\quad C[i] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\quad\quad C[A[j]] = C[A[j]] + 1$
6. $//C[i]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\quad\quad C[i] = C[i] + C[i-1]$
9. $//$ $C[i]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\quad\quad B[C[A[j]]] = A[j]$
12. $\quad\quad C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C\left[0..k\right]$ be a new array
2. **for** $i = 0$ **to** k
3. $\qquad C\left[i\right] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\qquad C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] + 1$
6. $//C\left[i\right]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\qquad C\left[i\right] = C\left[i\right] + C\left[i-1\right]$
9. $//\ C\left[i\right]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\qquad B\left[C\left[A\left[j\right]\right]\right] = A\left[j\right]$
12. $\qquad C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C[0..k]$ be a new array
2. **for** $i = 0$ **to** k
3. $\quad C[i] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\quad C[A[j]] = C[A[j]] + 1$
6. $//C[i]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\quad C[i] = C[i] + C[i-1]$
9. $//\ C[i]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\quad B[C[A[j]]] = A[j]$
12. $\quad C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
        $C[i] = 0$
for $j = 1$ **to** $A.length$
        $C[A[j]] = C[A[j]] + 1$
$//C[i]$ contains the number of elements equal to i
for $i = 1$ **to** k
        $C[i] = C[i] + C[i-1]$
$//$ $C[i]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
        $B[C[A[j]]] = A[j]$
        $C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
$//C[i]$ contains the number of elements equal to i
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
$//\ C[i]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C[0..k]$ be a new array
2. **for** $i = 0$ **to** k
3. $\quad C[i] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\quad C[A[j]] = C[A[j]] + 1$
6. $//C[i]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\quad C[i] = C[i] + C[i-1]$
9. $//$ $C[i]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\quad B[C[A[j]]] = A[j]$
12. $\quad C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C\left[0..k\right]$ be a new array
for $i = 0$ **to** k
    $C\left[i\right] = 0$
for $j = 1$ **to** $A.length$
    $C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] + 1$
$//C\left[i\right]$ contains the number of elements equal to i
for $i = 1$ **to** k
    $C\left[i\right] = C\left[i\right] + C\left[i-1\right]$
$//\ C\left[i\right]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
    $B\left[C\left[A\left[j\right]\right]\right] = A\left[j\right]$
    $C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

1. let $C\left[0..k\right]$ be a new array
2. **for** $i = 0$ **to** k
3. $\qquad C\left[i\right] = 0$
4. **for** $j = 1$ **to** $A.length$
5. $\qquad C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] + 1$
6. $//C\left[i\right]$ contains the number of elements equal to i
7. **for** $i = 1$ **to** k
8. $\qquad C\left[i\right] = C\left[i\right] + C\left[i-1\right]$
9. $//$ $C\left[i\right]$ contains the number of elements$\leq i$
10. **for** $j = A.length$ **to** 1
11. $\qquad B\left[C\left[A\left[j\right]\right]\right] = A\left[j\right]$
12. $\qquad C\left[A\left[j\right]\right] = C\left[A\left[j\right]\right] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
$//C[i]$ contains the number of elements equal to i
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
$//$ $C[i]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
$//C[i]$ contains the number of elements equal to i
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
$//$ $C[i]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Algorithm

Counting sort algorithm - **Assume indexing starting at 1**

Counting-Sort(A, B, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
$//C[i]$ contains the number of elements equal to i
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
$//$ $C[i]$ contains the number of elements$\leq i$
for $j = A.length$ **to** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Outline

Final Complexity

We have that

- Complexity $O(n + k)$.
- If $k = O(n)$, then the running time is$\Theta(n)$.

Final Complexity

We have that

- Complexity $O(n + k)$.
- If $k = O(n)$, then the running time is$\Theta(n)$.

Remarks

You can use it for

It is efficient if the range of input data is not significantly greater than the number of objects to be sorted.

It is STABLE

A sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

This why

It is often used as a sub-routine to another sorting algorithm like radix sort.

Remarks

You can use it for

It is efficient if the range of input data is not significantly greater than the number of objects to be sorted.

It is STABLE

A sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

This why

It is often used as a sub-routine to another sorting algorithm like radix sort.

Remarks

You can use it for

It is efficient if the range of input data is not significantly greater than the number of objects to be sorted.

It is STABLE

A sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

This why

It is often used as a sub-routine to another sorting algorithm like radix sort.

Outline

Introduction

Radix sort interesting facts!

- Radix sort is how IBM made its money, using punch card readers for census tabulation in early 1900's.
- A radix sorting algorithm was originally used to sort punched cards in several passes.

Introduction

Radix sort interesting facts!

- Radix sort is how IBM made its money, using punch card readers for census tabulation in early 1900's.
- A radix sorting algorithm was originally used to sort punched cards in several passes.

How?

It sorts each digit (or field/column) separately. Example:

$$\begin{matrix} 3 & 4 & \mathbf{5} & 1 \\ 1 & 2 & \mathbf{2} & 4 \\ 7 & 8 & \mathbf{9} & 1 \\ 1 & 2 & \mathbf{2} & 5 \end{matrix} \quad \Longrightarrow \quad \begin{matrix} 1 & 2 & \mathbf{2} & 4 \\ 1 & 2 & \mathbf{2} & 5 \\ 3 & 4 & \mathbf{5} & 1 \\ 7 & 8 & \mathbf{9} & 1 \end{matrix}$$

Introduction

Radix sort interesting facts!

- Radix sort is how IBM made its money, using punch card readers for census tabulation in early 1900's.
- A radix sorting algorithm was originally used to sort punched cards in several passes.

How?

It sorts each digit (or field/column) separately. Example:

$$\begin{array}{cccc} 3 & 4 & \mathbf{5} & 1 \\ 1 & 2 & \mathbf{2} & 4 \\ 7 & 8 & \mathbf{9} & 1 \\ 1 & 2 & \mathbf{2} & 5 \end{array} \Longrightarrow \begin{array}{cccc} 1 & 2 & \mathbf{2} & 4 \\ 1 & 2 & \mathbf{2} & 5 \\ 3 & 4 & \mathbf{5} & 1 \\ 7 & 8 & \mathbf{9} & 1 \end{array}$$

It starts with the least-significant digit

Radix sort must use a stable sort.

Introduction

Radix sort interesting facts!

- Radix sort is how IBM made its money, using punch card readers for census tabulation in early 1900's.
- A radix sorting algorithm was originally used to sort punched cards in several passes.

How?

It sorts each digit (or field/column) separately.Example:

$$\begin{array}{cccc} 3 & 4 & \mathbf{5} & 1 \\ 1 & 2 & \mathbf{2} & 4 \\ 7 & 8 & \mathbf{9} & 1 \\ 1 & 2 & \mathbf{2} & 5 \end{array} \Longrightarrow \begin{array}{cccc} 1 & 2 & \mathbf{2} & 4 \\ 1 & 2 & \mathbf{2} & 5 \\ 3 & 4 & \mathbf{5} & 1 \\ 7 & 8 & \mathbf{9} & 1 \end{array}$$

It starts with the least-significant digit

Radix sort must use a stable sort.

Introduction

Radix sort interesting facts!

- Radix sort is how IBM made its money, using punch card readers for census tabulation in early 1900's.
- A radix sorting algorithm was originally used to sort punched cards in several passes.

How?

It sorts each digit (or field/column) separately.Example:

$$\begin{matrix} 3 & 4 & \mathbf{5} & 1 \\ 1 & 2 & \mathbf{2} & 4 \\ 7 & 8 & \mathbf{9} & 1 \\ 1 & 2 & \mathbf{2} & 5 \end{matrix} \Longrightarrow \begin{matrix} 1 & 2 & \mathbf{2} & 4 \\ 1 & 2 & \mathbf{2} & 5 \\ 3 & 4 & \mathbf{5} & 1 \\ 7 & 8 & \mathbf{9} & 1 \end{matrix}$$

It starts with the least-significant digit

Radix sort must use a stable sort.

Outline

It is based in the following idea

First

Every number can be represented in each base. For example:

$$1024 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 \tag{2}$$

Thus in general, given a radix b

$$x = x_d b^d + x_{d-1} b^{d-1} + \ldots + x_0 b^0 \tag{3}$$

It can be proved inductively

We can sort by using Least-Significative to Most-significative Order and we keep an stable sort using this order

It is based in the following idea

First

Every number can be represented in each base. For example:

$$1024 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 \tag{2}$$

Thus in general, given a radix b

$$x = x_d b^d + x_{d-1} b^{d-1} + \ldots + x_0 b^0 \tag{3}$$

It can be proved inductively

We can sort by using Least-Significative to Most-significative Order and we keep an stable sort using this order

It is based in the following idea

First

Every number can be represented in each base. For example:

$$1024 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 \tag{2}$$

Thus in general, given a radix b

$$x = x_d b^d + x_{d-1} b^{d-1} + ... + x_0 b^0 \tag{3}$$

It can be proved inductively

We can sort by using Least-Significative to Most-significative Order and we keep an stable sort using this order

However

Remark 1

A most significant digit (MSD) radix sort can be used to sort keys in lexicographic order.

Remark 2

Unlike a least significant digit (LSD) radix sort, a most significant digit radix sort does not necessarily preserve the original order of duplicate keys.

However

Remark 1

A most significant digit (MSD) radix sort can be used to sort keys in lexicographic order.

Remark 2

Unlike a least significant digit (LSD) radix sort, a most significant digit radix sort does not necessarily preserve the original order of duplicate keys.

Example

Example

$$
\begin{array}{ccc}
3 & 2 & 9 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
8 & 3 & 9 \\
4 & 3 & 6 \\
7 & 2 & 0 \\
3 & 5 & 5
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & 2 & 0 \\
3 & 5 & 5 \\
4 & 3 & 6 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
3 & 2 & 9 \\
8 & 3 & 9
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & 2 & 0 \\
3 & 2 & 9 \\
4 & 3 & 6 \\
8 & 3 & 9 \\
3 & 5 & 5 \\
4 & 5 & 7 \\
6 & 5 & 7
\end{array}
\Longrightarrow
\begin{array}{ccc}
3 & 2 & 9 \\
3 & 5 & 5 \\
4 & 3 & 6 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
7 & 2 & 0 \\
8 & 3 & 9
\end{array}
$$

Example

Example

$$
\begin{array}{ccc}
3 & 2 & 9\\
4 & 5 & 7\\
6 & 5 & 7\\
8 & 3 & 9\\
4 & 3 & 6\\
7 & 2 & 0\\
3 & 5 & 5
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & 2 & \mathbf{0}\\
3 & 5 & \mathbf{5}\\
4 & 3 & \mathbf{6}\\
4 & 5 & \mathbf{7}\\
6 & 5 & \mathbf{7}\\
3 & 2 & \mathbf{9}\\
8 & 3 & \mathbf{9}
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & 2 & 0\\
3 & 2 & 9\\
4 & 3 & 6\\
8 & 3 & 9\\
3 & 5 & 5\\
4 & 5 & 7\\
6 & 5 & 7
\end{array}
\Longrightarrow
\begin{array}{ccc}
3 & 2 & 9\\
3 & 5 & 5\\
4 & 3 & 6\\
4 & 5 & 7\\
6 & 5 & 7\\
7 & 2 & 0\\
8 & 3 & 9
\end{array}
$$

Example

Example

$$
\begin{array}{ccc}
3 & 2 & 9 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
8 & 3 & 9 \\
4 & 3 & 6 \\
7 & 2 & 0 \\
3 & 5 & 5
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & 2 & \mathbf{0} \\
3 & 5 & \mathbf{5} \\
4 & 3 & \mathbf{6} \\
4 & 5 & \mathbf{7} \\
6 & 5 & \mathbf{7} \\
3 & 2 & \mathbf{9} \\
8 & 3 & \mathbf{9}
\end{array}
\Longrightarrow
\begin{array}{ccc}
7 & \mathbf{2} & 0 \\
3 & \mathbf{2} & 9 \\
4 & \mathbf{3} & 6 \\
8 & \mathbf{3} & 9 \\
3 & \mathbf{5} & 5 \\
4 & \mathbf{5} & 7 \\
6 & \mathbf{5} & 7
\end{array}
\Longrightarrow
\begin{array}{ccc}
3 & 2 & 9 \\
3 & 5 & 5 \\
4 & 3 & 6 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
7 & 2 & 0 \\
8 & 3 & 9
\end{array}
$$

Example

Example

$$
\begin{array}{ccc} 3 & 2 & 9 \\ 4 & 5 & 7 \\ 6 & 5 & 7 \\ 8 & 3 & 9 \\ 4 & 3 & 6 \\ 7 & 2 & 0 \\ 3 & 5 & 5 \end{array}
\Longrightarrow
\begin{array}{ccc} 7 & 2 & \mathbf{0} \\ 3 & 5 & \mathbf{5} \\ 4 & 3 & \mathbf{6} \\ 4 & 5 & \mathbf{7} \\ 6 & 5 & \mathbf{7} \\ 3 & 2 & \mathbf{9} \\ 8 & 3 & \mathbf{9} \end{array}
\Longrightarrow
\begin{array}{ccc} 7 & \mathbf{2} & 0 \\ 3 & \mathbf{2} & 9 \\ 4 & \mathbf{3} & 6 \\ 8 & \mathbf{3} & 9 \\ 3 & \mathbf{5} & 5 \\ 4 & \mathbf{5} & 7 \\ 6 & \mathbf{5} & 7 \end{array}
\Longrightarrow
\begin{array}{ccc} \mathbf{3} & 2 & 9 \\ \mathbf{3} & 5 & 5 \\ \mathbf{4} & 3 & 6 \\ \mathbf{4} & 5 & 7 \\ \mathbf{6} & 5 & 7 \\ \mathbf{7} & 2 & 0 \\ \mathbf{8} & 3 & 9 \end{array}
$$

Radix Sort: Algorithm Using the Least Significative Digit

Algorithm

Radix-Sort(A, d)

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

Outline

Iterative version using queues

How? First, assume Radix $= 10$

The integers are enqueued into an array of ten separate queues based on their digits from right to left.

Example: 170, 045, 075, 090, 002, 024, 802, 066

$$
\begin{array}{rc}
0: & 170, 190 \\
1: & none \\
2: & 002, 802 \\
3: & none \\
4: & 024 \\
5: & 045, 075 \\
6: & 066 \\
7: & none \\
8: & none \\
9: & none
\end{array}
$$

Iterative version using queues

How? First, assume Radix = 10

The integers are enqueued into an array of ten separate queues based on their digits from right to left.

Example: 170, 045, 075, 090, 002, 024, 802, 066

$$\begin{aligned} 0 &: \quad 17\mathbf{0}, 19\mathbf{0} \\ 1 &: \quad none \\ 2 &: \quad 00\mathbf{2}, 80\mathbf{2} \\ 3 &: \quad none \\ 4 &: \quad 02\mathbf{4} \\ 5 &: \quad 04\mathbf{5}, 07\mathbf{5} \\ 6 &: \quad 06\mathbf{6} \\ 7 &: \quad none \\ 8 &: \quad none \\ 9 &: \quad none \end{aligned}$$

Iterative version using queues

Then, the queues are dequeued back into an array of integers, in increasing order

$$170, 090, 002, 802, 024, 045, 075, 066$$

Then

You repeat again!!!

Iterative version using queues

Then, the queues are dequeued back into an array of integers, in increasing order

$$170, 090, 002, 802, 024, 045, 075, 066$$

Then

You repeat again!!!

Outline

Radix Sort: Proving the Complexity

Lemma 1

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n+k))$ time.

Proof

Quite simple!!!

Radix Sort: Proving the Complexity

Lemma 1

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n+k))$ time.

Proof

Quite simple!!!

Outline

Thanks Carlos Alcala Oracle Class 2014 for the Example

Imagine the following

That you have a sequence of n IP Addresses. For example:

$$n \text{ IP addresses} \begin{cases} 192.168.45.120 \\ 192.128.15.120 \\ 100.192.168.45 \\ \vdots \\ 92.16.4.120 \end{cases}$$

Thus, Why not use the chunks in the IP to sort the IP addresses (Think Columns)

$$n \text{ IP addresses} \begin{cases} 192.168.45.120 \\ 192.128.15.120 \\ 100.192.168.45 \\ \vdots \\ 92.16.4.120 \end{cases}$$

Thanks Carlos Alcala Oracle Class 2014 for the Example

Imagine the following

That you have a sequence of n IP Addresses. For example:

$$n \text{ IP addresses} \begin{cases} 192.168.45.120 \\ 192.128.15.120 \\ 100.192.168.45 \\ \vdots \\ 92.16.4.120 \end{cases}$$

Thus, Why not use the chunks in the IP to sort the IP addresses (Think Columns)

$$n \text{ IP addresses} \begin{cases} 192.168.\mathbf{45}.120 \\ 192.128.\mathbf{15}.120 \\ 100.192.\mathbf{168}.45 \\ \vdots \\ 92.16.\mathbf{4}.120 \end{cases}$$

Yes!!!

Yes!!!

- After all each chunk is a number between 0 and 255 i.e between 0 to $2^8 - 1 \Rightarrow$ size chunks is $r = 8$
- We have then for each IP address a size of $b = 32$ bits

Yes!!!

Yes!!!

- After all each chunk is a number between 0 and 255 i.e between 0 to $2^8 - 1 \Rightarrow$ size chunks is $r = 8$
- We have then for each IP address a size of $b = 32$ bits

This is a good example for the

Lemma 2

Given n b-bit numbers and any positive ingeter $r \leq b$, RADIX-SORT correctly sort these numbers in $\Theta(\frac{b}{r}(n+2^r))time.$

This is a good example for the

Lemma 2

Given n b-bit numbers and any positive ingeter $r \leq b$, RADIX-SORT correctly sort these numbers in $\Theta(\frac{b}{r}(n + 2^r))time.$

Proof

At the Board...

Final Remarks

Final Remarks

- LSD radix sorts have resurfaced as an alternative to high performance comparison-based sorting algorithms (like heapsort and mergesort) that require $O(n \log n)$ comparisons. **YES BIG DATA!!!**
- For more, look at V. J. Duvanenko, "In-Place Hybrid Binary-Radix Sort", Dr. Dobb's Journal, 1 October 2009

Final Remarks

Final Remarks

- LSD radix sorts have resurfaced as an alternative to high performance comparison-based sorting algorithms (like heapsort and mergesort) that require $O(n \log n)$ comparisons. **YES BIG DATA!!!**
- For more, look at V. J. Duvanenko, "In-Place Hybrid Binary-Radix Sort", Dr. Dobb's Journal, 1 October 2009

Outline

Assumptions

The keys are in the range $[0, 1)$

Actually you can have a range $[0, N)$ and divide the Keys by N

Assumptions

The keys are in the range $[0, 1)$

Actually you can have a range $[0, N)$ and divide the Keys by N

Something Notable

You have n of them

Then Create Cluster of them by using buckets/sets

$$\left\{ k | \text{if } \frac{i}{n} \leq k \wedge k < \frac{i+1}{n} \right\} \text{ with } i \in \{0, 1, 2, ..., n-1\} \tag{4}$$

Assumptions

The keys are in the range $[0, 1)$

Actually you can have a range $[0, N)$ and divide the Keys by N

Something Notable

You have n of them

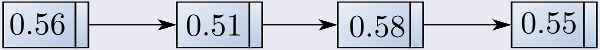
Then Create Cluster of them by using buckets/sets

$$\left\{ k | \text{if } \frac{i}{n} \leq k \wedge k < \frac{i+1}{n} \right\} \text{ with } i \in \{0, 1, 2, ..., n-1\} \tag{4}$$

What can we use to represent this sets?

We can use a link list for that

$\frac{5}{10} \leq$



$< \frac{6}{10}$

Then you have

A sequence of bucket to represent the sets

$$\left\{k \mid \text{if } \frac{i}{n} \leq k \wedge k < \frac{i+1}{n}\right\} \text{ with } i \in \{0, 1, 2, ..., n-1\} \tag{5}$$

What can we use to represent this sets?

We can use a link list for that

$\frac{5}{10} \leq$

$$0.56 \rightarrow 0.51 \rightarrow 0.58 \rightarrow 0.55$$

$< \frac{6}{10}$

Then you have

A sequence of bucket to represent the sets

$$\left\{ k | \text{if } \frac{i}{n} \leq k \wedge k < \frac{i+1}{n} \right\} \text{ with } i \in \{0, 1, 2, ..., n-1\} \tag{5}$$

What about this process?

Process

Create n linked lists (buckets) to divide interval $[0, 1)$ into subintervals of size $1/n$.

Thus

Add each input element to the appropriate bucket.

Then

Sort each list

Then

Add each input element to the appropriate bucket.

What about this process?

Process

Create n linked lists (buckets) to divide interval $[0, 1)$ into subintervals of size $1/n$.

Thus

Add each input element to the appropriate bucket.

Then

Sort each list

Then

Add each input element to the appropriate bucket.

What about this process?

Process

Create n linked lists (buckets) to divide interval $[0,1)$ into subintervals of size $1/n$.

Thus

Add each input element to the appropriate bucket.

Then

Sort each list

Then

Add each input element to the appropriate bucket.

What about this process?

Process

Create n linked lists (buckets) to divide interval $[0, 1)$ into subintervals of size $1/n$.

Thus

Add each input element to the appropriate bucket.

Then

Sort each list

Then

Add each input element to the appropriate bucket.

Outline

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\,[0..n-1]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4. make $B\,[i]$ an empty list
5. for $i = 0$ to n
6. insert $A\,[i]$ into list $B\,[\lfloor nA\,[i]\rfloor]$
7. for $i = 0$ to $n-1$
8. sort list $B\,[i]$ with insertion sort
9. concatenate the list $B\,[0], B\,[1], \ldots, B\,[n-1]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4. make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6. insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n-1$
8. sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4. make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6. insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n-1$
8. sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4. make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6. insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right] \rfloor\right]$
7. for $i = 0$ to $n-1$
8. sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4.   make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6.   insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n-1$
8.   sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4.   make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6.   insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n-1$
8.   sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n - 1$
4. make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6. insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n - 1$
8. sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right], B\left[1\right], \ldots, B\left[n-1\right]$ together in order

Bucket Sort Algorithm

Algorithm assuming

Buket-Sort(A)

1. let $B[0..n-1]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n - 1$
4. make $B[i]$ an empty list
5. for $i = 0$ to n
6. insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
7. for $i = 0$ to $n - 1$
8. sort list $B[i]$ with insertion sort
9. concatenate the list $B[0]$, $B[1]$, $\ldots$, $B[n-1]$ together in order

Bucket Sort Algorithm

Algorithm assuming

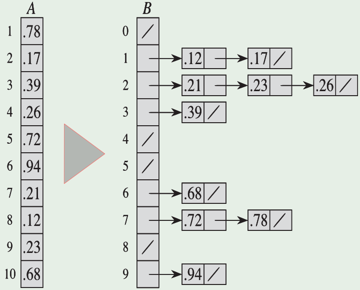
Buket-Sort(A)

1. let $B\left[0..n-1\right]$ be a new array
2. $n = A.length$
3. for $i = 0$ to $n-1$
4. make $B\left[i\right]$ an empty list
5. for $i = 0$ to n
6. insert $A\left[i\right]$ into list $B\left[\lfloor nA\left[i\right]\rfloor\right]$
7. for $i = 0$ to $n-1$
8. sort list $B\left[i\right]$ with insertion sort
9. concatenate the list $B\left[0\right]$, $B\left[1\right]$, ... , $B\left[n-1\right]$ together in order

Outline

Bucket Sort

Example



Outline

We have the following

We need to analyze the algoritm

But we have an insertion sort at each bucket!!!

Therefore

Let n_i the random variable on the size of the bucket.

We get the following complexity function

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O\left(n_i^2\right) \tag{6}$$

Look at the Board!!!

We have the following

We need to analyze the algoritm

But we have an insertion sort at each bucket!!!

Therefore

Let n_i the random variable on the size of the bucket.

We get the following complexity function

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O\left(n_i^2\right) \tag{6}$$

Look at the Board!!!

We have the following

We need to analyze the algoritm

But we have an insertion sort at each bucket!!!

Therefore

Let n_i the random variable on the size of the bucket.

We get the following complexity function

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O\left(n_i^2\right) \tag{6}$$

Look at the Board!!!

Final Complexity is

After using the expected value

$$E\left(T\left(n\right)\right)=\Theta\left(n\right)$$

Outline

Exercises

From Cormen's book solve the following

- 8.1-1
- 8.1-3
- 8.2-2
- 8.2-3
- 8.3-2
- 8.3-4
- 8.4-2