VE281
Data Structures and Algorithms

Non-comparison Sort

Announcement

- Programming Assignment One released
  - On sorting
  - Due time: 11:59 pm, Sep. 30th, 2017

Outline

- Non-comparison Sort
  - Counting Sort
  - Bucket Sort
  - Radix Sort

Counting Sort
A Simple Version

- Sort an array A of **integers** in the range $[0, k]$, where k is known.

1. Allocate an array `count[k+1]`.

2. Scan array A. For i`=1` to `N`, increment `count[A[i]]`.

3. Scan array `count`. For i`=0` to `k`, print `i` for `count[i]` times.

- Time complexity: $O(N + k)$.

- The algorithm can be converted to sort integers in some other known range $[a, b]$.
  - Minus each number by a, converting the range to $[0, b - a]$.

Counting Sort
A General Version

- In the previous version, we print `i` for `count[i]` times.
  - Simple but only works when sorting integer keys alone.
  - How to sort items when there is "additional" information with each key?
- A general version:
1. Allocate an array `C[k+1]`.
2. Scan array A. For i`=1` to `N`, increment `C[A[i]]`.
3. For `i=1` to `k`, `C[i]=C[i-1]+C[i]`
   - `C[i]` now contains number of items less than or equal to `i`.
4. For `i=N` downto `1`, put `A[i]` in new position `C[A[i]]` and decrement `C[A[i]]`.

Counting Sort
Example

k=5

1. Allocate an array **C[k+1]**.

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

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

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

3. For **i=1** to **k**, **C[i]= C[i-1]+C[i]**

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

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

6

Counting Sort
Example

k=5

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

1. Allocate an array **C[k+1]**.

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

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

1	2	3	4	5	6	7	8
						3	

3. For **i=1** to **k**, **C[i]= C[i-1]+C[i]**

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

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

Why putting 3 at location 7 is correct?

7

Counting Sort
Example

1. Allocate an array **C[k+1]**.

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

3. For **i=1** to **k**, C[i]= C[i-1]+C[i]

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

k=5

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

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

1	2	3	4	5	6	7	8
	0					3	

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

Counting Sort
Example

1. Allocate an array **C[k+1]**.

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

3. For **i=1** to **k**, C[i]= C[i-1]+C[i]

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

k=5

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

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

	1	2	3	4	5	6	7	8
		0				3	3	

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

Counting Sort
Example

k=5

1. Allocate an array **C[k+1]**.

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

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

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

3. For **i=1** to **k**, C[i]= C[i-1]+C[i]

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

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

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

Counting Sort
Example

1. Allocate an array `C[k+1]`.

2. Scan array A. For i=**1** to **N**, increment `C[A[i]]`.

3. For **i=1** to **k**, `C[i]= C[i-1]+C[i]`

4. For **i=N** downto **1**, put `A[i]` in new position `C[A[i]]` and decrement `C[A[i]]`.

k=5

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

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

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

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

Counting Sort
Example

1. Allocate an array **C[k+1]**.

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

3. For **i=1** to **k**, **C[i]= C[i-1]+C[i]**

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

k=5

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

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

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

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

Counting Sort
Example

1. Allocate an array **C[k+1]**.

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

3. For **i=1** to **k**, C[i]= C[i-1]+C[i]

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

k=5

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

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

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

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

13

Counting Sort
Example

1. Allocate an array **C[k+1]**.

2. Scan array A. For i**=1** to **N**, increment **C[A[i]]**.

3. For **i=1** to **k**, **C[i]= C[i-1]+C[i]**

4. For **i=N** downto **1**, put **A[i]** in new position **C[A[i]]** and decrement **C[A[i]]**.

k=5

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

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

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

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

Done!

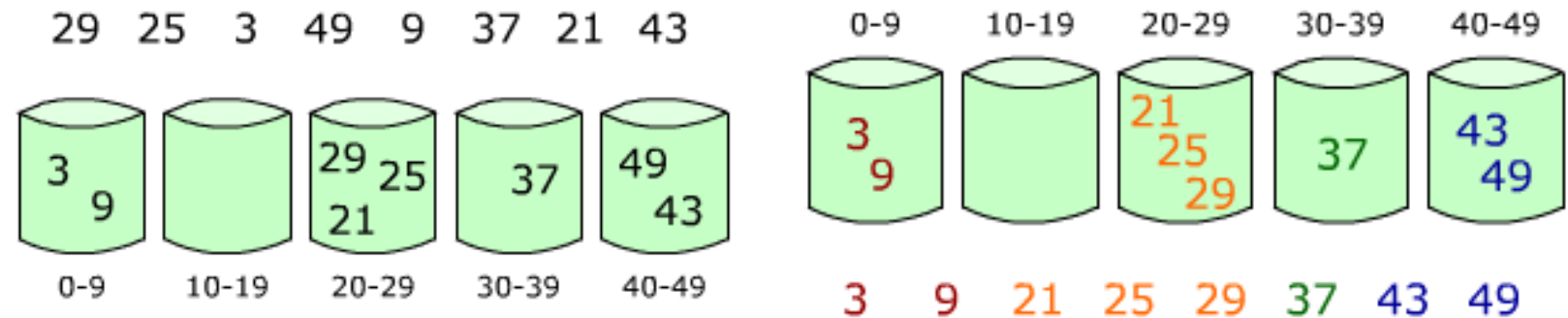Is counting sort stable?   Yes!

Outline

- Non-comparison Sort
  - Counting Sort
  - Bucket Sort
  - Radix Sort

Bucket Sort

- Instead of simple integer, each key can be a complicated record, such as a real value.

- Then instead of incrementing the count of each bucket, **distribute** the records **by their keys** into appropriate buckets.

- Algorithm:

1. Set up an array of initially empty "buckets".

2. Scatter: Go over the original array, putting each object in its bucket.

3. Sort each non-empty bucket <u>by a comparison sort</u>.

4. Gather: Visit the buckets in order and put all elements back into the original array.

Bucket Sort

- Example

29 25 3 49 9 37 21 43

0-9	10-19	20-29	30-39	40-49
3 9		29 25 21	37	49 43

0-9	10-19	20-29	30-39	40-49
3 9		21 25 29	37	43 49

3 9 21 25 29 37 43 49

- Time complexity
  - Suppose we are sorting cN items and we divide the entire range into N buckets.
  - Assume that the items are uniformly distributed in the entire range.
  - The average case time complexity is $O(N)$.

Outline

- Non-comparison Sort
  - Counting Sort
  - Bucket Sort
  - Radix Sort

Radix Sort

- **Radix sort** sorts integers by looking at one digit at a time.
- Procedure: Given an array of integers, from the least significant bit (LSB) to the most significant bit (LSB), repeatedly do **stable** bucket sort according to the current bit.

- For sorting base-b numbers, bucket sort needs b buckets.
  - For example, for sorting decimal numbers, bucket sort needs 10 buckets.

Radix Sort
Example

- Sort 815, 906, 127, 913, 098, 632, 278.

- Bucket sort 81$\underline{5}$, 90$\underline{6}$, 12$\underline{7}$, 91$\underline{3}$, 09$\underline{8}$, 63$\underline{2}$, 27$\underline{8}$ according to the least significant bit:

0	1	2	3	4	5	6	7	8	9
		63$\underline{2}$	91$\underline{3}$		81$\underline{5}$	90$\underline{6}$	12$\underline{7}$	09$\underline{8}$ 27$\underline{8}$	

- Bucket sort 6$\underline{3}$2, 9$\underline{1}$3, 8$\underline{1}$5, 9$\underline{0}$6, 1$\underline{2}$7, 0$\underline{9}$8, 2$\underline{7}$8 according to the second bit.

Radix Sort

Example

- Bucket sort 6<u>3</u>2, 9<u>1</u>3, 8<u>1</u>5, 9<u>0</u>6, 1<u>2</u>7, 0<u>9</u>8, 2<u>7</u>8 according to the second bit.

0	1	2	3	4	5	6	7	8	9
9<u>0</u>6	9<u>1</u>3 8<u>1</u>5	1<u>2</u>7	6<u>3</u>2				2<u>7</u>8		0<u>9</u>8

- Bucket sort <u>9</u>06, <u>9</u>13, <u>8</u>15, <u>1</u>27, <u>6</u>32, <u>2</u>78, <u>0</u>98 according to the most significant bit.

Radix Sort
Example

- Bucket sort <u>9</u>06, <u>9</u>13, <u>8</u>15, <u>1</u>27, <u>6</u>32, <u>2</u>78, <u>0</u>98 according to the most significant bit.

0	1	2	3	4	5	6	7	8	9
<u>0</u>98	<u>1</u>27	<u>2</u>78				<u>6</u>32		<u>8</u>15	<u>9</u>06 <u>9</u>13

- The final sorted order is: 098, 127, 278, 632, 815, 906, 913.

Radix Sort: Correctness

- <u>Claim</u>: after bucket sorting the i-th LSB, the numbers are sorted according to their last i digits

- Proof by mathematical induction

- Inductive step
  - Assume that according to the last i digits, order is $a_1 < \cdots < a_n$
  - For two adjacent numbers a_k and a_{k+1} if they are not in the same bucket, they are sorted according to their last i digits
  - If they are in the same bucket, then $a_k < a_{k+1}$ for the last $(i - 1)$ bits. They are also sorted due to stability of bucket sort

Radix Sort
Time Complexity

- Let k be the maximum number of digits in the keys and N be the number of keys.

- We need to repeat bucket sort k times.
  - Time complexity for the bucket sort is $O(N)$.

- The total time complexity is $O(kN)$.

Radix Sort

- Radix sort can be applied to sort keys that are built on **positional notation**.
  - **Positional notation**: all positions uses the same set of symbols, but different positions have different weight.
  - Decimal representation and binary representation are examples of positional notation.
  - Strings can also be viewed as a type of positional notation. Thus, radix sort can be used to sort strings.
- We can also apply radix sort to sort records that contain multiple keys.
  - For example, sort records (year, month, day).