Design and Analysis of Algorithm

Linear Time Sorting (Counting Sort)

Lecture -18



Overview

- Running time of counting sort is O(n+k).
- Required extra space for sorting.
- Is a stable sorting.

- Counting sort is a type of sorting technique which is based on keys between a specific range.
- It works by counting the number of objects having distinct key values (i.e. one kind of hashing).

- Consider the input set : 4, 1, 3, 4, 3. Then n=5 and k=4.
- Counting sort determines for each input element *x*, the number of elements less than *x*.
- This information is uses to place element *x* directly into its position in the output array.
- For example if there exits 17 elements less that x then x is placed into the 18th position into the output array.

Assumptions:

- *n* records
- Each record contains keys or data
- All keys are in the range of 0 to k, where k is the highest key value of the array.

• Space:

For coding this algorithm uses three array:

- **Input Array:** A[1..n] store input data, where n is the length of the array.
- **Output Array: B[1..***n***]** finally store the sorted data
- **Temporary Array:** C[0..k] store data temporarily

• Let us illustrate the counting sort with an example. Apply the concept of counting sort on the given array.

	1	2	3	4	5	6	7	8
Α	2	5	3	0	2	3	0	3

Let us illustrate the counting sort with an example.
 Apply the concept of counting sort on the given array.

 1
 2
 3
 4
 5
 6
 7
 8

 A
 2
 5
 3
 0
 2
 3
 0
 3

• First create a new array C[0....k], where k is the highest key value. And initialize with 0(i.e. zero)

for
$$i=0$$
 to k
 $C[i]=0;$

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

• Let us illustrate the counting sort with an example. Apply the concept of counting sort on the given array.

	1	2	3	4	5	6	7	8
Α	2	5	ന	0	2	3	0	ന

Find the frequencies of each object and store it in C array.

```
for j=1 to A. length
C[A[j]] = C[A[j]] + 1;
```

	0	1	2	3	4	5
С	2	0	2	3	0	1

Let us illustrate the counting sort with an example.
 Apply the concept of counting sort on the given array.

Find the frequencies of each object and store it in C array.

```
for j=1 to A. length
C[A[j]] = C[A[j]] + 1;
```

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

And then cumulatively add C array.

for
$$i=1$$
 to k
 $C[i] = C[i] + C[i-1];$

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
Α	2	5	3	0	2	3	0	3
,								i

	1	2	3	4	5	6	7	8
3							3	

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

for j=A. length down to 1 B[C[A[j]]] = A[j]; C[A[j]] = C[A[j]] - 1;

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

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

1 2 3 4 5 6 7 8 A 2 5 3 0 2 3 0 <mark>3</mark>

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
Α	2	5	3	0	2	3	0	3
·							j	

	1	2	3	4	5	6	7	8
В		0					3	

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

for j=A. length down to 1 B[C[A[j]]] = A[j]; C[A[j]] = C[A[j]] - 1;

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

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

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

	1	2	4	5	6	7	8
В		0				ര	

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

```
for j=A. length down to 1

B[C[A[j]]] = A[j];

C[A[j]] = C[A[j]] - 1;

A 2 5 3 0 2 3 0 3
```

	1	2	ന	4	5	6	7	8
В		0				3	3	

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
В		0				3	3	

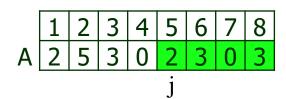
	1	2	3	4	5	6	7	8
В		0				3	3	

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

```
for j=A. length down to 1

B[C[A[j]]] = A[j];

C[A[j]] = C[A[j]] - 1;
```



	1	2	ന	4	5	6	7	8
В		0		2		3	3	

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
В		0		2		3	3	

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

```
for j=A. length down to 1 B[C[A[j]]] = A[j]; C[A[j]] = C[A[j]] - 1; A \ 2 \ 5 \ 3 \ 0 \ 2 \ 3 \ 0 \ 3 i
```

	1	2	3	4	5	6	7	8
В	0	0		2		3	3	

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

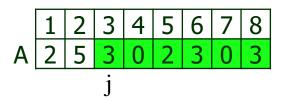
	1	2	3	4	5	6	7	8
В	0	0		2		3	3	

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

```
for j=A. length down to 1

B[C[ A[j] ]] = A[j];

C[ A[j] ] = C[ A[j] ] - 1;
```



	1	2	ന	4	5	6	7	8
В	0	0		2	3	3	3	

	0	1	2	3	4	5
С	0	2	3	5	7	8

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
В	0	0		2	3	3	3	

	1	2	3	4	5	6	7	8
В	0	0		2	3	3	3	

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

```
for j=A. length down to 1 B[C[A[j]]] = A[j]; C[A[j]] = C[A[j]] - 1; A \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 5 & 3 & 0 & 2 & 3 & 0 & 3 \\ & & & & & & & \\ \end{bmatrix}
```

	1	2	3	4	5	6	7	8
В	0	0		2	3	3	3	5

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
В	0	0		2	3	3	3	5

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

```
for j=A. length down to 1

B[C[A[j]]] = A[j];

C[A[j]] = C[A[j]] - 1;

A 2 5 3 0 2 3 0 :

i
```

	1	2	3	4	5	6	7	8
В	0	0	2	2	3	3	3	5

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

```
for j=A. length down to 1
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
```

	1	2	3	4	5	6	7	8
В	0	0	2	2	3	3	3	5

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

Counting-Sort(A, B, k)

- 1. Let C[0....k] be a new array
- 2. for i=0 to k
- 3. C[i] = 0;
- 4. for j=1 to A. length
- 5. C[A[j]] = C[A[j]] + 1;
- 6. for i=1 to k
- 7. C[i] = C[i] + C[i-1];
- 8. for j=A. length down to 1
- 9. B[C[A[j]]] = A[j];
- 10. C[A[j]] = C[A[j]] 1;

Counting-Sort(A, B, k)

- 1. Let C[0....k] be a new array
- 2. for i=0 to k
- 3. C[i] = 0;
- 4. for j=1 to A. length
- 5. C[A[j]] = C[A[j]] + 1;
- 6. for i=1 to k
- 7. C[i] = C[i] + C[i-1];
- 8. for j=A. length down to 1
- 9. B[C[A[j]]] = A[j];
- 10. C[A[j]] = C[A[j]] 1;

[Loop 1]

[Loop 2]

[Loop 3]

[Loop 4]

Complexity Analysis

Counting-Sort(A, B, k)

```
1. Let C[0....k] be a new array
```

2. for
$$i=0$$
 to k

[Loop 1]
$$O(k)$$
 times

3.
$$C[i] = 0$$
;

$$O(n)$$
 times

5.
$$C[A[j]] = C[A[j]] + 1;$$

6. for
$$i=1$$
 to k

[Loop 3]
$$O(k)$$
 times

7.
$$C[i] = C[i] + C[i-1];$$

8. for j=A. length down to 1 **[Loop 2]**
$$O(n)$$
 times

$$\mathbf{0}(n)$$
 times

9.
$$B[C[A[j]]] = A[j];$$

10.
$$C[A[j]] = C[A[j]] - 1;$$

Complexity Analysis

- So the counting sort takes a total time of: O(n + k)
- Counting sort is called stable sort.

(A sorting algorithm is *stable* when numbers with the same values appear in the output array in the same order as they do in the input array.)

Pro's and Con's of Counting Sort

- Pro's
 - Asymptotically very Fast O(n + k)
 - Simple to code
- Con's
 - Doesn't sort in place.
 - Requires O(n + k) extra storage space.

