L15 - Counting Sort

Thursday, May 14, 2020 7:52 AM

Non comparison based sorting

There are ***no comparisons among the elements of the list***.

Counting Sort assumes that each of its elements is in the range **0..k**, for some ***positive integer k***. If k is of the order of O(n), then it sorts in Linear Time.

In other words, Counting Sort is applicable only on positive integers.

For each element x of the list , it counts the number of x occurring in the list as well as the number of elements less than and equal to x.

It requires two additional arrays B[1..n] and C[0..k], where B is the output array and C keeps count of the number of elements that are less than each element in the list.

A 1 2 8

2_1	9	5_1	7	2_2	2_3	0	5_2

1. //Initialize the C array with zero
2. for i = 0 to k
3. C[i] = 0
4. //Count and store the number of each element in A
5. for j = 1 to A. length
6. C[A[j]] = C[A[j]] + 1

C

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

C

0	1	2	3	4	5	6	7	8	9
1	0	~~1 2~~ 3	0	0	~~1~~ 2	0	1	0	1

7. //Find the number of elements less than or equal to x
8. for j = 1 to k
9. C[j] = C[j] + C[j - 1]

C

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

```
for j = A. length downto 1    // for j=1 to A.length
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

A

2_1	9	5_1	7	2_2	2_3	0	5_2

B[C[A[j]]] = B[C[A[8]]] = B[C[5]] = B[6]

B

1	2	3	4	5	6	7	8
					5_2		

C

0	1	2	3	4	5	6	7	8	9
1	1	4	4	4	~~6~~ 5	6	7	7	8

B

1	2	3	4	5	6	7	8
0					5_2		

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	4	4	4	~~6~~ 5	6	7	7	8

B

1	2	3	4	5	6	7	8
0			2_3		5_2		

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ 3	4	4	~~6~~ 5	6	7	7	8

B

1	2	3	4	5	6	7	8
0		2_2	2_3		5_2		

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ ~~3~~ 2	4	4	~~6~~ 5	6	7	7	8

B

1	2	3	4	5	6	7	8
0		2_2	2_3		5_2	7	

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ ~~3~~ 2	4	4	~~6~~ 5	6	~~7~~ 6	7	8

B

1	2	3	4	5	6	7	8
0		2_2	2_3	5_1	5_2	7	

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ ~~3~~ 2	4	4	~~6~~ ~~5~~ 4	6	~~7~~ 6	7	8

B

1	2	3	4	5	6	7	8
0		2_2	2_3	5_1	5_2	7	9

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ ~~3~~ 2	4	4	~~5~~ ~~4~~ 3	6	~~7~~ 6	7	~~8~~ 7

B

1	2	3	4	5	6	7	8
0	2_1	2_2	2_3	5_1	5_2	7	9

C

0	1	2	3	4	5	6	7	8	9
~~1~~ 0	1	~~4~~ ~~3~~ ~~2~~ 1	4	4	~~5~~ ~~4~~ 3	6	~~7~~ 6	7	~~8~~ 7

Counting-Sort(A, B, k)

```
let C[0..k] be a dynamically allocated array of length k+1
//Initialize the C array with zero
 for i = 0 to k                          | O(k)
       C[i] = 0
//Count and store the number of each element in A
 for j = 1 to A. length                  | O(n)
       C[A[j]] = C[A[j]] + 1
//Find the number of elements less than or equal to x
for j = 1 to k                           | O(k)
     C[j] = C[j] + C[j - 1]
for j = A. length downto 1               | O(n)
     B[C[A[j]]] = A[j]
     C[A[j]] = C[A[j]] - 1
```

T(n) = O(n) + O(n) +O(k) +O(k) = 2c1n +2c2k = O(n + k)

N=10
K=1000

5 5 4 1
1 5 4 5
1 4 5 5

6 5 5 8
5 6 5 8
5 5 6 8

Stable Sort:
A Sorting algorithm is STABLE if the ***order of duplicate elements*** in the input is preserved in the sorted output.

Quick Sort	non-Stable
Merge Sort	Stable
Insertion Sort	Stable
Selection Sort	non-Stable
Bubble Sort	Stable