Design and Analysis of Algorithms

PART III

Dinesh Kullangal Sridhara
Pavan Gururaj Muddebihal

Counting Sort

Most of the algorithms cannot do better than O(nlogn). This algorithm assumes that each input element is in the range 0 to k for some integer k. The basic idea is to determine for each input element x, the number of elements less than x. This information is used to place the element x directly into its position in the output array.

Totally three arrays are involved.
-Input Array A[1….n]
-Sorted output array B[1…n]
-Temporary working array C[1…n].

For Ex. Consider the following Input array A.

A[1..8]

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

First the temporary array C is built as below such that each element C[i] contains the number of elements equal to i. Size of the array C is chosen based on the maximum element in A. [k=5]

For j←1 to length[A]
  Do C[A[j]]←C[A[j]] + 1

C[0..5]

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

C[0]=2 ➔there are 2 elements in A having value 0 i.e A[4],A[7].

C[1]=0 ➔there are no elements in A having value 1.
C[2]=2 ➔there are 2 elements in A having value 2 i.e A[1],A[5].
C[3]=3 ➔there are 3 elements in A having value 3 i.e A[3],A[6],A[8].
C[4]=0 ➔there are no elements in A having value 4
C[5]=1 ➔there is 1 element in A having value 5 i.e A[2].

Next C is reconstructed such that C[i] now contains the number of elements less than or equal to i.

For i← 1 to k
  Do C[i]←C[i]+C[i+1]

C[0..5]

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

C[0]=2 ➔there are 2 elements in A less than or equal to 0 .
C[1]=2 ➔there are 2 elements in A less than or equal to 1.
C[2]=4 ➔there are 4 elements in A less than or equal to 2
C[3]=7 ➔there are 7 elements in A less than or equal to 3.
C[4]=7 ➔there are 7 elements in A less than or equal to 4.
C[5]=8 ➔there is 8 element in A less than or equal to 5.

Finally every element of A is placed in its right position in the final sorted array B and we decrement the respective array value in C.

For j←length[A] downto 1
 Do B[C[A[j]]]->A[j]
 C[A[j]]← C[A[j]] - 1

A[8]=3 is placed at B[7] because the number of elements less than or equal to 3 is 7.i.e C[3]=7.

B[1..8]

1	2	3	4	5	6	7	8
						3	

Then decrement C[3].[7-1=6]

C[0..5]

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

Now A[7]=0 is placed at B[2] because number of elements less than or equal to 0 is 2 c[0]=2.

B[1..8]

1	2	3	4	5	6	7	8
	0					3	

Then decrement C[0].[2-1=1.]

C[0..5]

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

This is process is continued which gives us the final sorted array.

B[1..8]

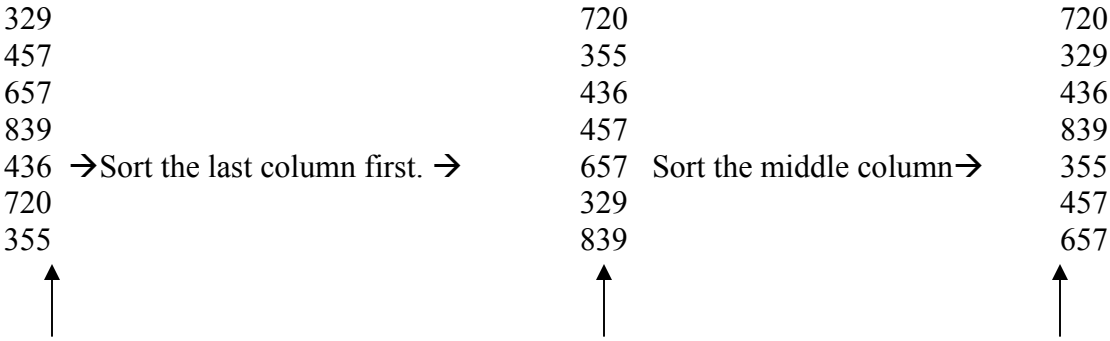
1	2	3	4	5	6	7	8
0	0	2	2	3	3	3	5

Running time

The overall time taken is $\Theta(k+n)$ as there are 2 for loops taking $\Theta(k)$ time and 2 for loops taking $\Theta(n)$ time.
Counting sort is usually used when $k=O(n)$ which gives a running time of $\Theta(n)$.

Radix Sort

Radix sort algorithm is used by the card sorting machines. Radix sort does sorting on the least significant digit first. The cards are then combined into a single deck with the cards in the 0 bin preceding the cards in the 1 bin preceding the cards in the 2 bin and so on. Then the entire deck is sorted again on the second least significant digit and recombined in the like manner. The process continues until the cards have been sorted on all d digits. At that point cards are fully sorted on the d digit number. D passes through the deck are required to sort. Following shows the operation of the radix sort on a deck of seven 3 digit numbers.



After sorting the 1st column we get the sorted array.

329
355
436
457
657
720
839

Hoare partition correctness

Hoare is the original partition algorithm, developed by T. Hoare. [P7.1 page 159]

Concept

Given an array,

- Two pointers are used, one pointing to the beginning of the array and the other pointing to the end of the array.
- The left pointer is moved to the right unless an element larger than the pivot is found.
- The right pointer is moved to the left unless an element smaller than the pivot element is found.
- The two elements pointed by the two the pointers are swapped.
- The procedure is repeated.
- The end condition for the procedure is that the two pointers meet each other.

Example

Input array A[] = 2 8 7 1 6 5 3 4
Pivot Selection = 4
Left Pointer P points to 2
Right Pointer Q points to 4

2	P 8	7	1	6	5	3	4 Q	Left Positioned
2	P 8	7	1	6	5	3 Q	4	Right Positioned
2	P 3	7	1	6	5	8 Q	4	After Swap
2	3	P 7	1	6	5	8 Q	4	Left Positioned
2	3	P 7	1Q	6	5	8	4	Right Positioned
2	3	P 1	7Q	6	5	8	4	After Swap
2	3	1	P 7Q	6	5	8	4	Left Positioned
2	3	1	P Q7	6	5	8	4	Pointers Collided
2	3	1	P Q4	6	5	8	7	Pivot Positioned

Output: 2 3 1 < 4 > 6 5 8 7

Pseudo Code

Hoare-partition (A, p, r)

```
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
        do repeat j ← j - 1
                until A[j] <= x
           repeat i ← i + 1
                until A[i] >= x
           if i < j
                then exchange A[i] ↔ A[j]
           else return j
```

Analysis:

- The analysis for Hoare partition is similar to the partition implementation using two pointers starting from same side.
- Number of comparisons is N, the number of elements.
- The number of exchanges is the same as the original implementation.
- Worst case (n^2).
- Expected case is (nlogn)

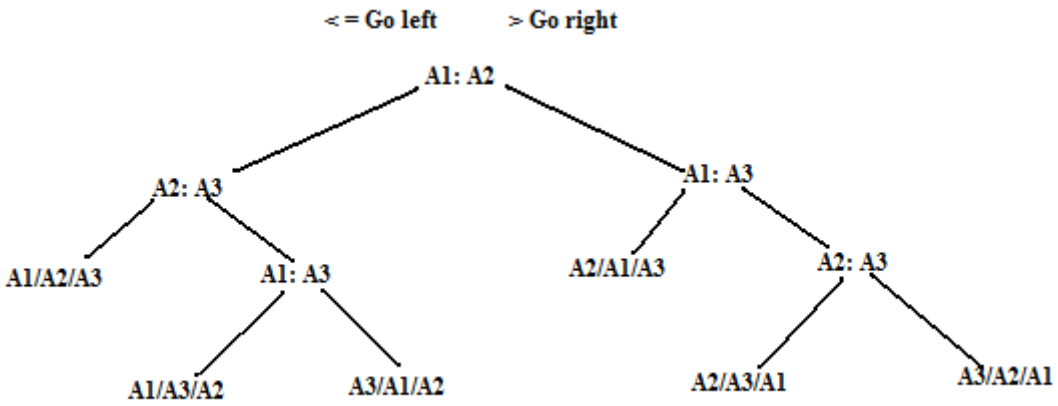
The decision-tree Model

A decision tree is a full binary tree representing the comparisons between elements performed by a particular sorting algorithm operating on an input of a given size.

Example Decision Tree

Consider the relation <= and >. The notation i:j represents the comparison between elements Ai and Aj. Relation used: If Ai <= Aj then move left else move right

The decision tree for above mentioned conditions is given below



Such a tree with outcomes as leaves and decisions as internal nodes may be constructed for an algorithm and a specific value of n [the number of inputs]

Observations

- Worst case number of comparisons for a given comparison sort algorithm = Height of decision tree
- Any comparison sort algorithm requires Ω (nlogn) comparisons in the worst time
- Since there must be !n leaves, the minimum height of decision tree for sorting n keys is Ω (log(!n)) = Ω (nlogn).