Radix Sort

Manoj Gupta

September 18, 2016

In the last lecture, we saw that the worst case running time of *any* comparison based sorting algorithm is $n \log n$. So, can we come up with an algorithm which is not comparison based. In general then answer is no. However we may be able to do something if the range of the input is known. Assume that you have n where each number is in the range $[1 \dots 999]$. Note that n may be less than 1000. We give an $O(n)$ time algorithm to sort these numbers. To this end, we make a new auxiliary array B os size 1000. We can view B to represent a set which can contain only elements from the range $[1 \dots n]$. Initially $B[i] = 0$ indicating that i is not the part of the set. We now use the following algorithm:

```
make a new array $B[1 \dots 999]$;
for $i \leftarrow 1$ to $n$ do
    $B[A[i]] \leftarrow 1$;
end
for $i \leftarrow 1$ to 999 do
    if $B[i] \leftarrow 1$ then
        print "$i$" ;
    end
end
```

The first for loop takes $O(n)$ time while the time taken by the second for loop is proportional to the size of the array. One can check that the above algorithm sorts the array A. However, the running time of the above algorithm is not $O(n)$ because it is proportional to the size of B.

However, if we were to sort these digits just based on the most significant digit (assume that each number is represented by three digit, so 0 = 000, 10 = 010 etc.), then this can be done by using an array $B[0 \dots 9]$ of size 10. When we see an element with most significant digit i, then we put it in cell i. However, this presents us with a problem because the i^{th} cell might already have some element. So, we keep a linked list in each cell. When we want to add an element in cell i, we put it at the end of the linked list at cell i.

```
make a new array $B[0 \dots 9]$;
for $i \leftarrow 1$ to $n$ do
    if the $j^{th}$ digit of $A[i]$ is $k$ then
        Add $A[i]$ to the linked list in cell $k$
    end
end
Add the elements to $A$ from left to right;
for $i \leftarrow 0$ to 9 do
    Add all the elements in the linked list in cell $i$ to the start of the array $A$;
end
```

Figure 1: BUCKETSORT(A, j)

This algorithm is called BUCKETSORT and it just sorts A based on digit j. One can check that the time taken by the above algorithm is $O(n)$. We are now ready to use the above algorithm to sort an array A which contains only integers from $[1 \dots 999]$.

```
BUCKETSORT$(A, d)$;
Assume that all the elements in $A$ with elements $i(0 \leq i \leq 9)$ are in the range $n_i$ to $n_{i+1} - 1$ such
  that $n_0 = 1$ and $n_{10} - 1 = n$;
for $i \leftarrow 0$ to 9 do
    RADIXSORT$(A[n_i \dots n_{i+1} - 1], d - 1)$;
end
```

Figure 2: RADIXSORT$(A[1 \dots n], d)$

The following observation is clear from the BUCKETSORT algorithm.

Observation 0.1. *After* BUCKETSORT(A, d)*, A contains numbers that are sorted based on the d^{th} digit.*

We now show that RADIXSORT will sort the array. Consider two number abc and def in array A. We will now show the following lemma:

Lemma 0.2. *After* RADIXSORT *completes, the final position of abc in A is smaller than the final position of def, if abc $< def$.*

Proof. There are three case:

1. $a < d$

   In this case, the first iteration of RADIXSORT will call BUCKETSORT$(A, 3)$ will put abc at a smaller index than def. After this we partition the array based on their first digit and then call RADIXSORT on each of these sub-arrays. Within these sub-arrays, the number can change their position but number cannot be swapped across sub-arrays. This is due to the fact that all the numbers in the bucket i start will digit i and these number can only lie in the array A from index n_i to $n_{i+1} - 1$. Thus, in the subsequent invocations of RADIXSORT, def will find its final place in $[n_d \dots n_{d+1} - 1]$ and abc will find its final place in $[n_a \dots n_{a+1} - 1]$. Since $a < d$, $n_{a+1} - 1 < n_d$. Thus, the final place of abc is less than the final place of def.

2. $a = d$ and $b < e$.

   In this case, in the first call to RADIXSORT($A[1 \dots n]$,3), both abc and def will be part of the same bucket. However, in the second iteration, abc will be a part of the b^{th} bucket and def will be the part of the e^{th} bucket. Since $b < e$, we claim that the position of abc is less than the position of def after the second iteration. As in case (1), we can again show that the relative position of abc and def don't change after this.

3. $a = d$, $b = e$ and $c < f$

   Same as the case above.

□

Thus after RADIXSORT completes, each pair of elements in the array satisfy Lemma 0.2. We claim that the array is sorted in the above lemma is true for all elements in the array. This can be proved using induction and the reader is encouraged to do it.

We now move on to prove the running time of the above algorithm. Let $T(n, d)$ be the running time of RADIXSORT$(A[1 \dots n], d)$. In this procedure, we partition the array into at most 9 sub-array and then call radix sort on each of these sub-array. The time taken to partition is the the time taken by BUCKETSORT(A, d) which is proportional to the size of the array, that is $O(n)$. Also, we assume in procedure RADIXSORT that numbers with the first digit i lie in $[n_i \dots n_{i+1} - 1]$ index of the array. Thus, we can calculate the running time of RADIXSORT as follows:

$T(n, d) = \sum_{i=0}^{9} T(n_{i+1} - n_i, d - 1) + cn$

$T(n, 1) = cn$

where the base case says that the time taken to sort n numbers with one digit is $O(n)$. This can be done by BUCKETSORT.

We now prove that $T(n) \leq cdn$ using induction

Lemma 0.3. $T(n) \leq cdn$

Proof. We will prove by induction. For the base case, $T(n, 1)$ is the time taken by BUCKETSORT to sort n single digit numbers.

Assume that $T(n, d - 1) \leq c(d - 1)n$. We will now prove that $T(n, d) = cdn$. We will use the recurrence relation: $T(n, d) = \sum_{i=0}^{9} T(n_{i+1} - 1 - n_i, d - 1) + cn$. Using induction hypothesis. $T(n, d) \leq \sum_{i=0}^{9} c(n_{i+1} - n_i)(d - 1) + cn = c(n^{10} - 1 - n_0)(d - 1) + cn$. Since $n^{10} - 1 = n$ and $n_0 = 1$, we get: $T(n) \leq c(n - 1)(d - 1) + cn \leq cnd$. □

1 RADIXSORT from least significant bit to most significant bit

In the previous section, we saw an algorithm that sorts the input from most significant digit to the least significant digit. In this section, we will see another algorithm that sorts from least significant digit to most significant digit. This sorting is also the simplest – in terms of coding and analysis – and also beautiful in my opinion.

```
1 $d \leftarrow$ number of digits in the input;
2 foreach $k \leftarrow 1$ to $d$ do
3     Sort $A$ in a stable way looking at $k$-th digit only;
4 end
```

Figure 3: RADIXSORT($A[1 \ldots n]$)

Few comments are in order: (1) This is not a recursive function. (2) The sorting has to be done in a *stable* way. We next define stable sorting:

Definition 1.1. *A sorting algorithm is called stable if two elements having the same value maintain their relative order after the algorithm sorts the array. That is, if there exists two indices in the unsorted array, i and j such that $A[i] = A[j]$, then their final position in the sorted array i' and j' satisfy $i' < j'$.*

The reader can verify that INSERTION SORT, MERGESORT and BUCKETSORT (explained at the start of this write-up) are stable sorting algorithm while HEAPSORT and QUICKSORT are not a stable sorting algorithm. Let us see the run of our new RADIXSORT algorithm on the following input.

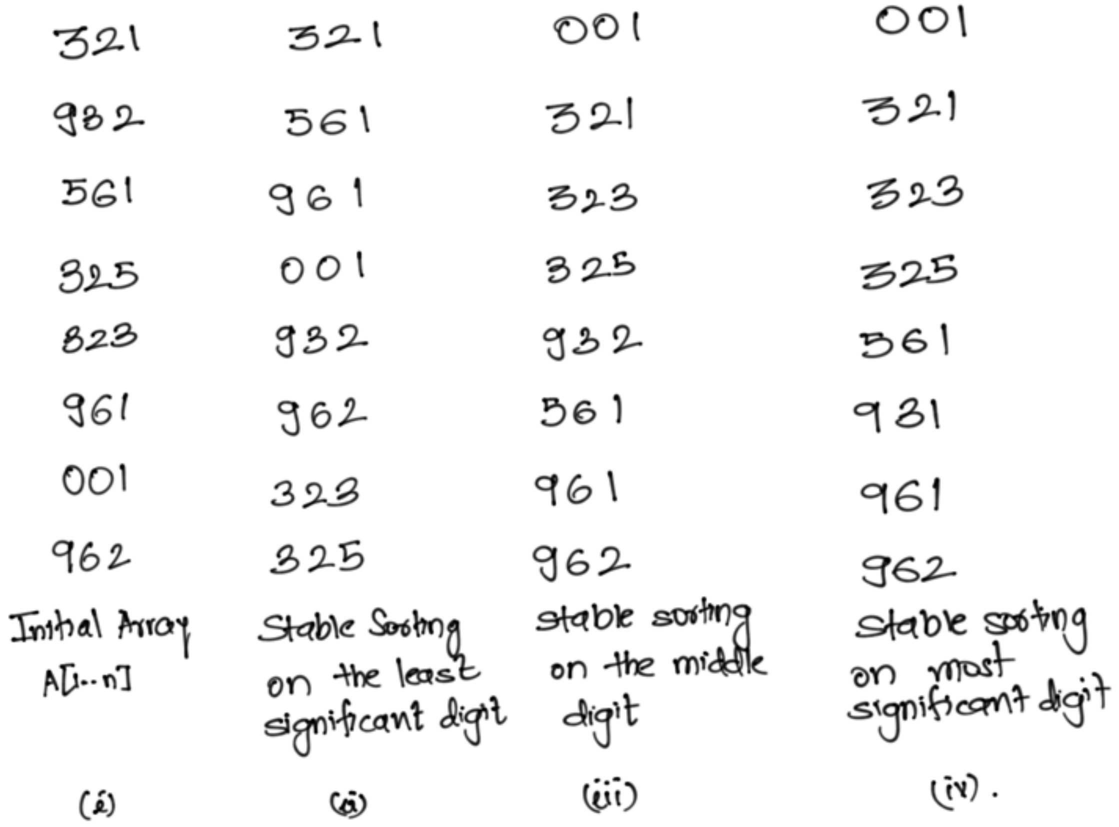

Figure 4: (i) The initial array. (ii) Array after stable sorting on MSD. (iii) Array after sorting on the middle digit. (iv) The final sorted array.

We now show that the above algorithm correctly sorts the array:

Lemma 1.2. *After* RADIXSORT *completes, the final position of abc in A is smaller than the final position of def, if $abc < def$.*

Proof. Again there are three cases:

1. $a < d$

   No matter what happens in the first two iteration of the algorithm, in the third iteration, we claim that the final position of abc will be less than def. This is due to the fact that in the third iteration, we are sorting based on the most significant digit and $a \ < d$.

2. $a = d$ and $b < e$

   No matter what happens in the first iteration, after the second iteration, we claim that position of abc is less than the position of def. And in the third iteration, since our sorting is stable, our algorithm will place abc at a smaller index than def (since at the start of the third iteration, the index of abc is less than def and $a = d$).

3. $a = d$ and $b = e$ and $c < f$.

   Similar to case (2).

□

Thus, using Lemma 1.2, we can again claim that the array is sorted after RADIXSORT ends. For calculating the running time, note that we can use sc BucketSort to sort the array based on k-th digit in the k-th iteration. Since BUCKETSORT is a stable sorting algorithm and takes $O(n)$ time, the total time taken by our algorithm is $O(nd)$ if each number has d digits.