Sorting Methods III

Andrew C. Lee

EECS, Syracuse

Non Comparision Based Sorting Methods

Some examples:

1. Counting Sort
2. Radix Sort
3. Bucket Sort
4. Reading: CLRS Chapter 8 and Chapter 9

Note the assumption(s) each of these methods made

Counting Sort: Ideas

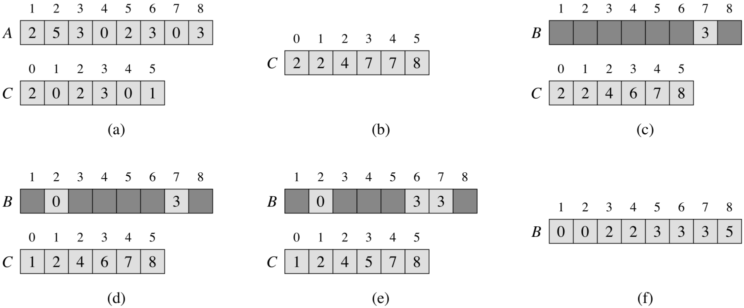

Figure : How Counting Sort Works

Counting Sort: Pseudocode

COUNTING-SORT(A, B, n, k)

let $C[0..k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** n
    $C[A[j]] = C[A[j]] + 1$
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
for $j = n$ **downto** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Figure : Pseudocode: Counting Sort

Worst Case Complexity: $\Theta(k + n)$ time

Radix Sort: Idea

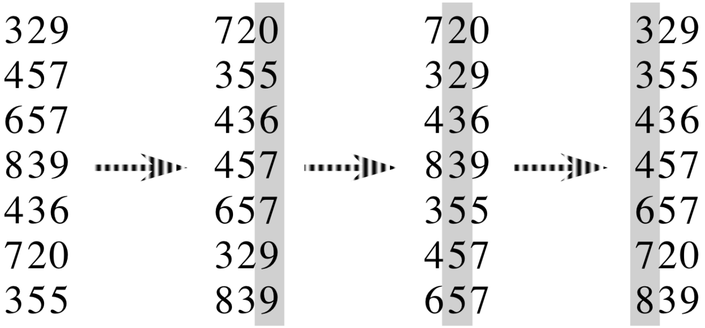

Figure : Radix Sort: Ideas

Radix Sort: Pseudocode and Complexity

```
RADIX-SORT(A, d)
for i = 1 to d
    use a stable sort to sort array A on digit i
```

Lemma 8.3

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n+k))$ time if the stable sort it uses takes $\Theta(n+k)$ time.

Figure : Radix Sort Results

Bucket Sort: Idea

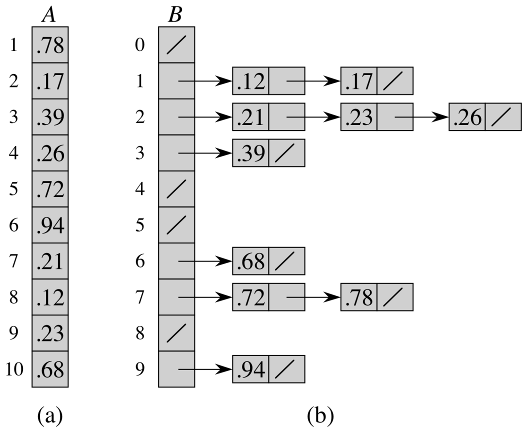

Figure : Bucket Sort: Ideas

Bucket Sort: Pseudocode and Complexity

```
BUCKET-SORT(A)
let B[0..n-1] be a new array
n = A.length
for i = 0 to n-1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n-1
    sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ..., B[n-1] together in order
```

Figure : Pseudocode   Note: *Average Case Running Time*: $O(n)$

Selection Problems

Discussions: Can we improve quick sort so that the worst case running time is $O(n \lg n)$?

What if we can *select* the **median** as *pivot* each time when we perform the partition step ?

The Select algorithm: Ideas

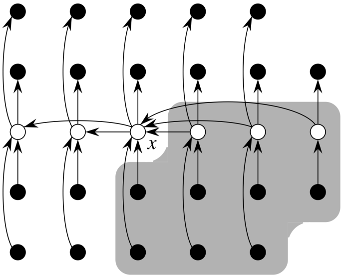

Figure : The ideas behind the algorithm Select

The Select algorithm: Sketches of Pseudocode

The SELECT algorithm determines the ith smallest of an input array of $n > 1$ distinct elements by executing the following steps. (If $n = 1$, then SELECT merely returns its only input value as the ith smallest.)

1. Divide the n elements of the input array into $\lfloor n/5 \rfloor$ groups of 5 elements each and at most one group made up of the remaining n mod 5 elements.
2. Find the median of each of the $\lceil n/5 \rceil$ groups by first insertion-sorting the elements of each group (of which there are at most 5) and then picking the median from the sorted list of group elements.
3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2. (If there are an even number of medians, then by our convention, x is the lower median.)
4. Partition the input array around the median-of-medians x using the modified version of PARTITION. Let k be one more than the number of elements on the low side of the partition, so that x is the kth smallest element and there are $n-k$ elements on the high side of the partition.
5. If $i = k$, then return x. Otherwise, use SELECT recursively to find the ith smallest element on the low side if $i < k$, or the $(i-k)$th smallest element on the high side if $i > k$.

Figure : The algorithm Select: Main Steps