SIENAcollege
Computer Science

Computer Science 385
Analysis of Algorithms
Siena College
Spring 2011

Topic Notes: Sorting by Counting

We have considered several sorting algorithms to this point: bubble sort, selection sort, merge sort, insertion sort, quicksort, tree sort and heapsort. We will not briefly consider a couple more.

Comparison Counting Sort

Consider this approach to the sorting problem: for each element in the collection to be sorted, count the number of elements that are smaller than that element. Once we have all of these counts, the count corresponding to each element is its position in the sorted array.

```
comparison_counting_sort(A[0..n-1])
  // S[0..n-1] is the sorted array
  // counts[0..n-1] is the array of counts of smaller elements for each
  counts[0..n-1] = 0
  for i = 0 to n-2
    for j = i+1 to n-1
      if A[i] < A[j] counts[j]++
      else counts[i]++
  for i = 0 to n-1
    S[count[i]] = A[i]
  return S
```

This algorithm is clearly $\Theta(n^2)$ and even uses $\Theta(n)$ of extra space, so it does not seem to be a leading candidate for use before our other algorithms. The idea does work well, however, in some cases...

Distribution Counting Sort

Suppose the set of elements we are to sort are known to come only from a small set of values. Perhaps just A's and B's. If we count the total number of A's in the input (call it n_A), then fill in the first n_A elements of the sorted array with A's, and the rest with B's, we've sorted the input!

The generalization of this idea is known as *distribution counting*, where we have a known range of values from a lower bound to an upper bound in the input array.

```
distribution_counting(A[0..n-1],lower,upper)
  // D[0..u-l] is the array of distributions/frequencies
  // S[0..n-1] is the sorted array
```

```
D[0..u-l] = 0
for i = 0 to n-1
  D[A[i]-l]++ // increment frequency of elt at A[i]
for j = 1 to u-l
  D[j] = D[j-1] + D[j]  // convert to distribution
for i = n-1 to 0
  j = A[i] - l
  S[D[j] - 1] = A[i]
  D[j]--
return S
```

What about the efficiency class of this algorithm? We have no nested loops or recursive calls! This is clearly linear. So it's easily the most efficient sorting algorithm we have encountered. But...it has the very significant restriction that we need extra space proportional to the range of values. So if we do have a limited range of valyes, this is a very good option.