Non-Comparison Based Sorts

COMP 215 Lecture 15
Radix Sort

• For the sorts today we take advantage of the fact that we know something about the items to be sorted.

• Radix sort:
  – We know that the items are positive integers represented in base 10 (or some other base).
  – Basic idea is to sort the items one digit at a time.
  – First impulse would be to start at the left and work to the right.
  – Let's give it a try...
Radix Sort

- It works, but there is quite a bit of bookkeeping involved.
- Turns out there is a better way.
- Starting with the ones place, put each item into one of ten bins.
  - This amounts to sorting by the rightmost digit.
- Repeat the process for each digit working from right to left.
- As long as we don't change the relative ordering of keys in the same bin, we have a correct sort.
- Let's see an example...
Radix Implementation

- We can implement radix sort using linked lists:
- Start by placing all keys into a master list.
- For each digit working right to left
  - Place each key at the end of the appropriate list for the value of the current digit.
  - Iterate through the ten lists from 0 to 9 placing each item in each list at the end of the master list.
- At the end, the master list will contain the sorted keys.
Radix Analysis

- Dealing with each digit requires $\Theta(n)$ time.
- Input can have an arbitrary number of digits $d$.
- Overall running time is $\Theta(dn)$.
- If $d$ is a constant, then this is $\Theta(n)$.
- Discussion Question...
Bucket Sort

• Bucket sort is useful if we know that are keys are uniformly distributed in the range \([0, 1)\).
• We create \(n\) “buckets”, each of which may store a list of items.
• If the keys are in array \(A\), we place each key into the bucket indexed \(\lfloor nA[i] \rfloor\).
• Then sort contents of each bucket using selection sort.
• Worst case performance? Can we improve it?
• Average case performance turns out to be \(\Theta(n)\).
  – We won't do the detailed analysis. Key is that the expected number of keys per bucket is 1.