EECS 3101 A: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101A`
Also on Moodle

Linear-time Sorting Algorithms

- The $\Omega(n \log n)$ lower bound is for comparison-based sorting algorithms
- We can do better than the lower bound if the algorithm is not comparison-based
- We can sort using information other than comparisons between data items if we restrict the scope of the problem
- For some restricted scenarios, we can sort in worse-case linear time

Bucket Sort

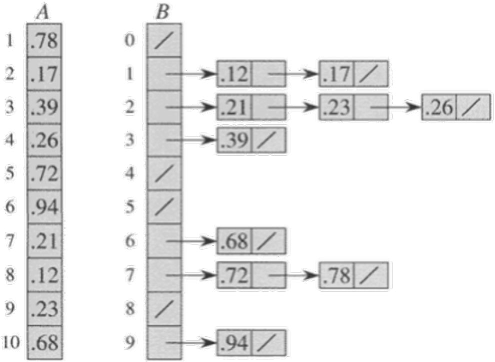
- Suppose all keys come from a finite interval, say $[0, 1)$
- Suppose you have 10 keys
- We can define buckets for ranges, e.g. $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1)$
- Insert keys in appropriate bucket
- If input is random and uniformly distributed, expected number of keys in each bucket is 1.

Bucket Sort - 2

- Suppose all keys come from a finite interval, say $[0, 1)$
- Suppose you have n keys
- Divide $[0, 1)$ into n equal-sized subintervals (buckets)
- Insert the n numbers into buckets
- Sort numbers in each bucket (insertion sort as default).
- Then go through buckets in order, listing elements
- If input is random and uniformly distributed, expected run time is $\Theta(n)$.

Bucket Sort - 3

- So given $A[1..n]$, create new array B of length n
- Insert $A[i]$ into $B[\lfloor nA[i] \rfloor]$



Bucket Sort: Properties and Extensions

- Stable Sort
- Keys must be numbers – since they are used to generate array indices
- Extension: Set of fixed keys like the set of names of 50 US states – Sort the keys and give each key its unique bucket. Insert each item into the bracket corresponding to its key
- What if input numbers are NOT uniformly distributed?
- What if the distribution is not known a priori?
- Can we get worst-case linear time algorithms?

Counting Sort

- applies when the keys come from a finite (and preferably small) set, e.g., are integers in the range $[0 \ldots k-1]$, for some fixed integer k
- We can then create an array $V[0 \ldots k-1]$ and use it to count the number of elements with each key $0 \ldots k-1$
- Then each input element can be placed in exactly the right place in the output array in constant time
- How to do this?
    - Determine the number of elements less than x, for each input x
    - Place x directly in its position

Counting Sort - pseudocode

```
COUNTINGSORT(A, B, k)
for i = 0 to k
        C[i] = 0
for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
// C[i] contains number of elements equal to i.
for i = 1 to k
        C[i] = C[i] + C[i − 1]
// C[i] contains number of elements ≤ i
for j = A.length downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] − 1
```

Counting Sort: Analysis and Comments

- Total cost is $\Theta(k+n)$, suppose $k = O(n)$, then total cost is $\Theta(n)$.
- So it beats the $\Omega(n \log n)$ lower bound
- Counting Sort is stable
- Q: can counting sort be used to sort large integers efficiently?

Radix Sort

- Input: An array of n numbers, each containing d digits
- Output: Sorted array
- Approach: Each digit (column) can be sorted (e.g., using Counting Sort)
- Q: Which digit to start from?

Generalization: Each "digit" between 0 and $k-1$ (inclusive)

Radix Sort - 2

- Sort the numbers using the least significant digit
- Sort by the next least significant digit
- Are the last 2 columns sorted?
- Generalize: after j iterations, the last j columns are sorted

Radix Sort - 3

1019	2231	1019	1019	1019
3075	3075	2225	3075	2225
2225	2225	2231	2225	2231
2231	1019	3075	2231	3075

Sorted!

1019	1019
3075	2231
2231	2225
2225	3075

Not sorted!

Radix Sort - 4

RADIXSORT(A, d)

for $i = 1$ **to** d
    use a stable sort to sort A on digit i

- Analysis: Given n d-digit numbers where each digit takes on up to k values, Radix-Sort sorts these numbers correctly in $\Theta((d(n+k))$ time.
- Loop invariant: Before iteration i, the keys have been correctly stable-sorted with respect to the $i-1$ least-significant digits

Radix Sort - Questions

- Sorting postal codes (e.g. the postal code for York is "M3J 1P3")
- Assume that you have to sort n numbers in the range 0 to $n^2 - 1$. Describe how you can use radix sort to sort them in $O(n)$ time.