CSE 202—Day 6

Linear sorting and order statistics

Neil Rhodes

UC San Diego
Overview

- Linear Sorting
- Order statistics
Linear Sorting

Linear sorting is possible because we have more information about the keys other than that they can be compared (in fact, some of the algorithms never compare items)

- Count sorting
- Radix sorting
- More probability
- Bucket sorting
Count Sorting

**Input:** Array $A[1..n]$ of integers $\in 1..k$ for some $k$

**Output:** Array $B[1..n]$, containing contents of $A$, but in sorted order.

**Runtime:** $O(k + n)(O(n) \text{ if } k = O(n))$

**Stable:** yes

**Idea:** Don’t ever compare; just count the number of occurrences of each integer.
Count Sorting

Overview of the algorithm

- Create an array, $C[1..k]$ and count the number of occurrences of each integer $\in 1..k$ in $A$
- Modify $C$ to contain the number of occurrences $\leq$ each integer $k$
- Go through $A$ backwards. For each element $e$
  - Use $C[e]$ to find how many elements are $\leq e$. That’s its location in $B$
  - Decrement $C[e]$ so that duplicates of $e$ aren’t placed on top of $e$
Count Sorting

Runtime

- $O(k)$ to initialize $C$
- $O(n)$ to count occurrences of items in $A$
- $O(k)$ to update $C$ from $=$ to $\leq$
- $O(n)$ to create all elements in $B$

Total: $O(n + k)$
Radix Sorting

How punch cards were sorted when sorters could sort only on one column

**Input:** Array $A[1..n]$ of $d$-digit numbers. Each digit has $k$ possible values

**Output:** Array $B[1..n]$ containing contents of $A$ but in sorted order

**Runtime:** $O(d(k+n))$

**Stable** yes

**Idea:** Use a stable sort to sort on *least-significant* digit, then next, up to the most-significant digit
Radix Sorting

Algorithm (Let digit 1 be the least significant digit)

\[ i = 1 \]

loop

LI: A is sorted on all digits < i

exit when \( i > d \)

B = StableSortOnIthDigit(A, i)

A = B

i = i + 1

end loop

postcondition: A is sorted on all digits \( \leq d \)
Radix Sorting

Conclusion

Which stable sort to use? Counting sort is a good choice

Why is a stable sort important?

Runtime Analysis: Counting sort takes $O(n + k)$ for each of $d$ times. Total = $O(d(n + k))$. If $d$ is constant, takes $O(n + k)$. 
More probability

*Variance* measures variability of a random variable

- **Example**
  - $X =$ total shown on a fair die. $E[X] = 3.5$
  - Roll a fair die. If it comes up 6, $Y$ is 21, otherwise, $Y$ is 0. $E[Y] = 3.5$
  - Variance is defined as $E[(X - E[X])^2]$
    (Equivalently, $Var[X] = E[X^2] - E^2[X]$)

  - $Var[X] = \frac{(1 - 3.5)^2 + (2 - 3.5)^2 + (3 - 3.5)^2 + (4 - 3.5)^2 + (5 - 3.5)^2 + (6 - 3.5)^2)}{6} = 8.75$

  - $Var[Y] = \frac{(5 \cdot (0 - 3.5)^2 + (21 - 3.5)^2)}{6} = 61.25$
More probability

**Bernoulli Trial:** Experiment which has only two possible outcomes: success (with probability $p$) and failure (with probability $q = 1 - p$). In a sequence, each trial is mutually independent.

**Binomial Distribution:** Random variable $X =$ number of successful experiments in $n$ Bernoulli trials

- $Pr\{X = k\} = \binom{n}{k} p^k q^{n-k}$
- $E[X] = np$
- $Var[X] = npq$
More probability

Continuous uniform probability distribution:
Over a closed interval \([a, b]\), we want points equally likely. We can’t have non-zero probabilities for an infinite number of points, though, so we define, for \([c, d]\) (a sub-interval of \([a..b]\))

\[
Pr\{[c, d]\} = \frac{d - c}{a - b}
\]
Bucket sort

Input: Array $A[1..n]$ of values drawn from a continuous uniform probability distribution in $[0..1)$.

Output: Array $B[1..n]$ containing contents of $A$ but in sorted order

Runtime: $O(n)$ average case

Stable: yes

Idea: Make $n$ buckets representing each of $n$ equal sub-intervals of $[0..1)$. Put each element in corresponding bucket. Then, sort each bucket.
**Bucket sort**

**Code**

Create Buckets[1..n], an array of empty linked lists.
for i = 1 to n
    Insert A[i] in linked list Buckets[ceil(n*A[i])]
Assert: if i < j, elements of Buckets[i] < elements of Buckets[j]
for i = 1 to n
    Sort linked list Buckets[n*A[i]] using insertion sort
Concatenate all elements from Buckets into B
Bucket sort

Analysis of average-case runtime (Expectation over all possible inputs)

- Putting elements in Buckets: \( O(n) \)
- Iterating over each bucket, \( \sum_{i=1}^{n} O(n_i^2) \) where \( n_i \) is length of bucket \( i \)
- Concatenating: \( O(n) \)
Bucket sort

Expected time

\[ E[T(n)] = E[O(n) + \sum_{i=1}^{n} O(n_i^2)] \]

\[ = O(n) + \sum_{i=1}^{n} O(E[n_i^2]) \]
Bucket sort

Analyzing $E[n_i^2]$ (different analysis from CLRS)

$$E[n_i^2] = Var[n_i] + E^2[n_i]$$

$$= npq + (np)^2$$

Binomial distribution

$$= n(1/n)((n-1)/n) + (n/n)^2$$

$$= \frac{n-1}{n} + 1$$

$$= 2 - 1/n$$
Bucket sort

Expected time

\[ E[T(n)] = E[O(n) + \sum_{i=1}^{n} O(n_i^2)] \]

\[ = O(n) + \sum_{i=1}^{n} O(E[n_i^2]) \]

\[ = O(n) + \sum_{i=1}^{n} O(2 - 1/n) \]

\[ = O(n) + n \times O(2 - 1/n) \]

\[ = O(n) \]
Order statistics

Order statistics are a generalization of min, max, and median.

- $i$-th order statistic of a set is the $i$th smallest element
- Minimum is first order statistic
- Maximum is $n$th order statistic
- Median is $(n + 1)/2$-th order statistic (round down if even number of elements)
Randomized order statistic

Input: Array $A[1..n]$ of distinct numbers and a number $k \in [1..n]$

Output: $k$-th smallest value of $A$.

Runtime: Expected time $O(n)$

Idea: Use Randomized Partition. If exactly $k - 1$ elements in left partition, return partition value. Otherwise recurse into appropriate partition.
Randomized order statistic

Stupid version mimicking Quicksort

Select(A, k)
    pick x in A uniformly at random
    partition into L1<x, L2=x, L3>x
    Quicksort(L1);
    Quicksort(L3);
    Concatenate L1, L2, L3
    return k’th element in concatenation

Note that if k’th element is in L1, sorting of L3 not needed and if k’th element is in L3, sorting of L1 not needed
Randomized order statistic

Select(A, k)

    pick x in A uniformly at random
    partition into L1<x, L2=x, L3>x
    if k < Length(L1) then
        return Select(L1, k)
    else if k > Length(L1) + Length(L2) then
        return Select(L3, k - Length(L1) - Length(L2))
    else
        return x
Randomized order statistic

Runtime analysis:
Worst-case: $O(n^2)$
Non-rigorous Average-case analysis:

- We end up with a recurrence on $n$ and $k$. Worst-case happens to be when $k = n/2$ so let’s use that case.
- Every call, we remove $|L2| + \min(|L1|, |L3|)$ items.
- Partition value equally likely to be $z_1, z_2, \ldots, z_n$ (using earlier terminology)
- $T(n) = n - 1 + \sum_{i=n/2}^{n} \frac{2}{n} T(i)$
Randomized order statistic

Solve by substitution. Assume $\exists c$ such that $T(i) \leq ci$ for $i < n$

$$T(n) = n - 1 + \sum_{i=n/2}^{n} \frac{2}{n}T(i) \leq n - 1 + \sum_{i=n/2}^{n} \frac{2}{n}ci$$

$$= n - 1 + \frac{2c}{n} \sum_{i=n/2}^{n} i = n - 1 + \frac{2c}{n} \left( \sum_{i=1}^{n} i - \sum_{i=1}^{n/2-1} i \right)$$

$$= n - 1 + \frac{2c}{n} \left( \frac{n^2}{2} - \frac{n^2}{8} + O(n) \right)$$

$$= n - 1 + \frac{3}{4}cn + O(1) = cn - \frac{cn}{4} + n + O(1)$$

If $c > 4$ (and $n_0$ big enough to swamp $O(1)$ term).
Deterministic order statistic

**Input:** Array $A[1..n]$ of distinct numbers and a number $k \in [1..n]$

**Output:** $k$-th smallest value of $A$.

**Runtime:** $O(n)$

**Idea:** Find a splitter (recursively) that’s guaranteed to cause a constant fraction of items to be ruled out
Deterministic order statistic

Input: Array $A[1..n]$ of distinct numbers and a number $k \in [1..n]$

Output: $k$-th smallest value of $A$.

Runtime: $O(n)$

Idea: Find a splitter (recursively) that’s guaranteed to cause a constant fraction of items to be ruled out
**Deterministic order statistic**

Select(A, k)

Line up elements of A in groups of 5
For each group, find median, x[i]
medianOfMedians = Select(x, Length(x)/2)
Use medianOfMedians to partition A
Call Select recursively on appropriate partition
Deterministic order statistic

Runtime analysis:

- medianOfMedians is greater than 1/2 the $x_i$s (1/10 of A)
- Each $x_i$ is greater than 2 other elements in its group
- medianOfMedians is greater than those 2 elements (2/10 of A)
- medianOfMedians is greater than 3/10 of the elements in A
- Similarly, medianOfMedians is less than 3/10 of the elements in A
- Recurrence $T(n) = O(n) + T(n/5) + T(7n/10)$
Deterministic order statistic

Runtime analysis:

\[ T(n) = \sum_{i=0}^{\log_{7} n} \left( \frac{9}{10} \right)^{i} cn = O(n) \]