More Sorting; Searching

Dan Barrish-Flood
Bucket Sort

• Put keys into n buckets, then sort each bucket, then concatenate.

• If keys are uniformly distributed (*quite an assumption*), then each bucket will be small, so each bucket can be sorted very quickly (using probably *insertion sort*).

• Analysis is complex, because this involves probability and random distribution (see CLRS p. 175 – 176!!)

• Bucket sort runs in $\Theta(n)$, i.e. linear!
Bucket Sort, cont’d

Assume the keys are in the range [0,1)

Divide the range [0,1) into n equal-sized buckets, so that the 1\textsuperscript{st} bucket is [0, 1/n)

the 1\textsuperscript{st} bucket is [0, 1/n)

the 2\textsuperscript{nd} bucket is [1/n , 2/n)

....

the n\textsuperscript{th} bucket is [(n-1)/n , 1)

If the buckets are B[0], B[1], ..., B[n-1] then the appropriate bucket for the key A[i] is FLOOR(n A[i])
Bucket Sort Code

```plaintext
BUCKET-SORT(A)
1  n ← length[A]
2  for i ← 1 to n
3      do insert A[i] into list B[[n A[i]]]
4  for i ← 0 to n - 1
5      do sort list B[i] with insertion sort
6  concatenate the lists B[0], B[1], ..., B[n - 1]
```
Bucket Sort in Action
When Linear Sorts can be used

<table>
<thead>
<tr>
<th></th>
<th>Integer</th>
<th>Real</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counting Sort</td>
<td>Yes*</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Radix Sort</td>
<td>Yes</td>
<td>No</td>
<td>Yes*</td>
</tr>
<tr>
<td>Binsort (=BucketSort)</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes*</td>
</tr>
</tbody>
</table>

*with modifications
The Selection Problem

Input: a set $A$ of $n$ (distinct) numbers, and a number $i$ where $1 \leq i \leq n$

Output: the element $x \in A$ that is larger than exactly $i-1$ other elements of $A$ (in practice, we allow ties).

The $i$th smallest value is called the $i$th order statistic.

The 1st order statistic is the minimum.

The $n$th order statistic is the maximum

the $n/2$ order statistic is the median (OK, there can be two medians, but we usually go with the lower one).

Q. Solve the selection problem in $\Theta(?)$ time...
Min and Max

Can find the min (or max) in *n-1 comparisons* in the way you learned in your first programming class:

MINIMUM(A)
1. min ← A[1]
2. **for** i ← 2 to `length[A]`
3. **do if** min > A[i]
4. **then** min ← A[i]
5. **return** min

The n-1 is also a lower bound....
Min and Max (Cont’d)

...thus we can find both min and max in $2n - 2$ comparisons. But we can do a bit better (though not asymptotically better). We can find both min and max in $3 \cdot \text{floor}(n/2)$ comparisons:

• keep track of both min and max
• for each pair of elements:
  ➢ first compare them to each other;
  ➢ compare the larger to max
  ➢ compare the smaller to min

This is the lower bound.
Selection in General

RANDOMIZED-PARTITION(A, p, r)
1. i ← RANDOM(p, r)
2. exchange A[r] with A[i]
3. return PARTITION(A, p, r)

An average-case \( \Theta(n) \) algorithm for finding the \( i \)th smallest element:
like QuickSort, but only recurse on one part.

\[
\text{RANDOMIZED-SELECT}(A, p, r, i)
\begin{align*}
1 & \quad \text{if } p = r \\
2 & \quad \quad \text{then return } A[p] \\
3 & \quad q ← \text{RANDOMIZED-PARTITION}(A, p, r) \\
4 & \quad k ← q - p + 1 \\
5 & \quad \text{if } i = k \quad \triangleright \text{the pivot value is the answer} \\
6 & \quad \quad \text{then return } A[q] \\
7 & \quad \text{elseif } i < k \\
8 & \quad \quad \text{then return } \text{RANDOMIZED-SELECT}(A, p, q - 1, i) \\
9 & \quad \text{else return } \text{RANDOMIZED-SELECT}(A, q + 1, r, i - k)
\end{align*}
\]

Average-case analysis along the same lines as quicksort. Worst-case running time?
Solution in Linear time (Worst-case)

A clever algorithm of largely theoretical interest.

SELECT(A, i) finds the $i$th smallest element of A in $\Theta(n)$ time, worst-case.

Idea: partition, but do some extra work to find a split that is guaranteed to be good.
the SELECT algorithm
More on SELECT

n = 5: o  o  o  o  o  o
find the median directly, or by sorting

n = 25:
1. make 5 groups of 5 each
2. find the median of each group (that’s 5 numbers)
3. find the median-of-medians, x.

note: x > about half the elements of about half the groups, and x < about half the elements of about half the groups. So x is in the middle half of the set (not necessarily the exact middle element!), so x is a great pivot for PARTITION. To put it more precisely...
SELECT high-level description

SELECT(A, i)

1. Divide A into floor(n/5) groups of 5 elements (and possibly one smaller groups of left-overs, as in the figure in an earlier slide).

2. Find the median of each 5-element group. Put these ceiling(n/5) medians into an array B.

3. //find the median-of-medians
   x ← SELECT(B, (1/2)ceiling(n/5))

4. PARTITION A around x, where k is the number of elements on the low side, so that x is the kth smallest element.

5. if i = k then return x, otherwise call 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 otherwise.
How many elements are greater than $x$?

There are $\left\lceil \frac{n}{5} \right\rceil$ groups.

At least $\frac{1}{2}$ contribute 3 elements $> x$, except for the group containing $x$ and the left - over group.

So "number of elements $> x"

\[ \geq 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \]

\[ \geq \frac{3}{10} n - 6 \]

Same argument shows that at least

\[ \frac{3}{10} n - 6 \] elements are $< x$.  

So the largest possible size of a partition is

\[ n - \left( \frac{3}{10} n - 6 \right) \]

\[ = \frac{7}{10} n + 6 \]
Recurrence for SELECT

\[ T(n) \leq \Theta(1) \quad \text{if } n \leq 80 \]

\[ T(n) \leq T\left(\left\lfloor \frac{n}{5} \right\rfloor \right) + T\left(\frac{7n}{10} + 6\right) + O(n) \quad \text{if } n > 80 \]

Proof by substitution (i.e. induction)

Assume \( T(n) \leq cn \) for some \( c \) and all \( n \leq 80 \)

\[
T(n) \leq C\left\lfloor \frac{n}{5} \right\rfloor + C\left(\frac{7n}{10} + 6\right) + O(n)
\]

\[
\leq \frac{cn}{5} + \frac{7cn}{10} + 6c + O(n)
\]

\[
= \frac{9cn}{10} + 7c + O(n)
\]

We need \( cn \geq \frac{9cn}{10} + 7c + O(n) \), or

\[
c\left(\frac{n}{10} - 7\right) \geq O(n)
\]

We can pick a large enough \( c \) when \( n > 80 \), thus SELECT is \( \Theta(n) \), worst-case