Design and Analysis of Algorithms

CSE 5311
Lecture 9  Median and Order Statistics

Junzhou Huang, Ph.D.
Department of Computer Science and Engineering
Medians and Order Statistics

• The $i$th order statistic of $n$ elements $S=\{a_1, a_2, \ldots, a_n\}$: $i$th smallest elements
• Also called selection problem
• Minimum and maximum
• Median, lower median, upper median
• Selection in expected/average linear time
• Selection in worst-case linear time
Order Statistics

- The *ith* order statistic in a set of *n* elements is the *ith* smallest element.
- The *minimum* is thus the 1st order statistic.
- The *maximum* is (duh) the *n*th order statistic.
- The *median* is the *n*/2 order statistic.
  - If *n* is even, there are 2 medians.
- *How can we calculate order statistics?*
- *What is the running time?*
Order Statistics

• **How many comparisons are needed to find the minimum element in a set? The maximum?**

• **Can we find the minimum and maximum with less than twice the cost?**

• Yes:
  – Walk through elements by pairs
    ➢ Compare each element in pair to the other
    ➢ Compare the largest to maximum, smallest to minimum
  – Total cost: 3 comparisons per 2 elements = \( O(3n/2) \)
**$O(n \lg n)$ Algorithm**

- Suppose $n$ elements are sorted by an $O(n \lg n)$ algorithm, e.g., MERGE-SORT
  - Minimum: the first element
  - Maximum: the last element
  - The $i$th order statistic: the $i$th element.
  - Median:
    - If $n$ is odd, then $(\frac{n+1}{2})$th element.
    - If $n$ is even,
      - then $(\lfloor \frac{n+1}{2} \rfloor)$th element, lower median
      - then $(\lceil \frac{n+1}{2} \rceil)$th element, upper median
- All selections can be done in $O(1)$, so total: $O(n \lg n)$.
- Can we do better?
Selection in Expected Linear Time $O(n)$

- Select $i$th element
- A divide-and-conquer algorithm RANDOMIZED-SELECT
- Similar to quicksort, partition the input array recursively
- Unlike quicksort, which works on both sides of the partition, just work on one side of the partition.
  - Called prune-and-search, prune one side, just search the other side).
Finding Order Statistics: The Selection Problem

• A more interesting problem is selection: finding the $i$th smallest element of a set

• We will show:
  – A practical randomized algorithm with $O(n)$ expected running time
  – A cool algorithm of theoretical interest only with $O(n)$ worst-case running time
Randomized Selection

- **Key idea**: use `partition()` from quicksort
  - But, only need to examine one subarray
  - This savings shows up in running time: $O(n)$

- **We will again use a slightly different partition than the book**:
  
  $q = \text{RandomizedPartition}(A, p, r)$

  \[
  p \quad \leq A[q] \quad \quad \quad \geq A[q] \quad \quad r
  \]
Randomized Selection

```
RandomizedSelect(A, p, r, i)
    if (p == r) then return A[p];
    q = RandomizedPartition(A, p, r)
    k = q - p + 1;
    if (i == k) then return A[q];
    if (i < k) then
        return RandomizedSelect(A, p, q-1, i);
    else
        return RandomizedSelect(A, q+1, r, i-k);
```

\[
\begin{array}{c|c|c}
\leq A[q] & \text{ } & \geq A[q] \\
\hline
p & q & r
\end{array}
\]
Randomized Selection

- Analyzing `RandomizedSelect()`
  - Worst case: partition always 0:n-1
    \[ T(n) = T(n-1) + O(n) = ??? \]
    \[ = O(n^2) \] (arithmetic series)
    
    - No better than sorting!
  - “Best” case: suppose a 9:1 partition
    \[ T(n) = T(9n/10) + O(n) = ??? \]
    \[ = O(n) \] (Master Theorem, case 3)
    
    - Better than sorting!
    
    - What if this had been a 99:1 split?
Randomized Selection

- **Average case**
  - For upper bound, assume $i$th element always falls in larger side of partition:
    \[
    T(n) \leq \frac{1}{n} \sum_{k=0}^{n-1} T(\max(k-1, n-k)) + \Theta(n)
    \]
  - Let’s show that $T(n) = O(n)$ by substitution

What happened here?

\[
\leq \frac{2}{n} \sum_{k=n/2}^{n-1} T(k) + \Theta(n)
\]

- Let’s show that $T(n) = O(n)$ by substitution

Max($k-1, n-k$) = $k-1$ if $k > \lceil n/2 \rceil$
Max($k-1, n-k$) = $n-k$ if $k \leq \lceil n/2 \rceil$
Randomized Selection

- Assume $T(n) \leq cn$ for sufficiently large $c$:

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

The recurrence we started with

$$\leq \frac{2}{n} \sum_{k=n/2}^{n-1} c k + \Theta(n)$$

Substitute $T(n) \leq cn$ for $T(k)$

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

“Split” the recurrence

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

Expand arithmetic series

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

Multiply it out
Randomized Selection

- Assume $T(n) \leq cn$ for sufficiently large $c$:

\[
T(n) \leq c(n - 1) - \frac{c}{2} \left( \frac{n}{2} - 1 \right) + \Theta(n)
\]

The recurrence so far

\[
= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n)
\]

Multiply it out

\[
= cn - \left( \frac{cn}{4} + \frac{c}{2} - \Theta(n) \right)
\]

Subtract $c/2$

\[
\leq cn \quad \text{(if } c \text{ is big enough)}
\]

Rearrange the arithmetic

What we set out to prove
Worst-Case Linear-Time Selection

• Randomized algorithm works well in practice
• What follows is a worst-case linear time algorithm, really of theoretical interest only
• Basic idea:
  – Generate a good partitioning element
  – Call this element $x$
Worst-Case Linear-Time Selection

• The algorithm in words:
  1. Divide \( n \) elements into groups of 5
  2. Find median of each group \((\text{How? How long?})\)
  3. Use \texttt{Select()} recursively to find median \( x \) of the \( \lfloor n/5 \rfloor \) medians
  4. Partition the \( n \) elements around \( x \). Let \( k = \text{rank}(x) \)
  5. \textbf{if} \( (i == k) \) \textbf{then} return \( x \)
     \textbf{if} \( (i < k) \) \textbf{then} use \texttt{Select()} recursively to find \( i \)th smallest element in first partition
     \textbf{else} \( (i > k) \) use \texttt{Select()} recursively to find \( (i-k) \)th smallest element in last partition
Worst-Case Linear-Time Selection

• (Sketch situation on the board)

• **How many of the 5-element medians are ≤ x?**
  – At least 1/2 of the medians = \( \left\lfloor \frac{n}{5} \right\rfloor / 2 = \left\lfloor \frac{n}{10} \right\rfloor \)

• **How many elements are ≤ x?**
  – At least 3 \( \left\lfloor \frac{n}{10} \right\rfloor \) elements

• For large \( n \), \( 3 \left\lfloor \frac{n}{10} \right\rfloor \geq n/4 \) (How large?)

• So at least \( n/4 \) elements ≤ \( x \)

• Similarly: at least \( n/4 \) elements ≥ \( x \)
Worst-Case Linear-Time Selection

• Thus after partitioning around $x$, step 5 will call Select() on at most $3n/4$ elements

• The recurrence is therefore:

$$T(n) \leq T(\lfloor n/5 \rfloor) + T(3n/4) + \Theta(n)$$

$$\leq T(n/5) + T(3n/4) + \Theta(n) \quad \lfloor n/5 \rfloor \leq n/5$$

$$\leq cn/5 + 3cn/4 + \Theta(n) \quad \text{Substitute } T(n) = cn$$

$$= 19cn/20 + \Theta(n) \quad \text{Combine fractions}$$

$$= cn - (cn/20 - \Theta(n)) \quad \text{Express in desired form}$$

$$\leq cn \quad \text{if } c \text{ is big enough} \quad \text{What we set out to prove}$$
Worst-Case Linear-Time Selection

• Intuitively:
  – Work at each level is a constant fraction (19/20) smaller
    ➢ Geometric progression!
  – Thus the O(n) work at the root dominates
Linear-Time Median Selection

• Given a “black box” $O(n)$ median algorithm, what can we do?
  – $i$th order statistic:
    ✓ Find median $x$
    ✓ Partition input around $x$
    ✓ if $(i \leq (n+1)/2)$ recursively find $i$th element of first half
    ✓ else find $(i - (n+1)/2)$th element in second half
    ✓ $T(n) = T(n/2) + O(n) = O(n)$
  – *Can you think of an application to sorting?*
Linear-Time Median Selection

- **Worst-case** $O(n \lg n)$ quicksort
  - Find median $x$ and partition around it
  - Recursively quicksort two halves
  - $T(n) = 2T(n/2) + O(n) = O(n \lg n)$
Summary

• The $i$th order statistic of $n$ elements $S=\{a_1, a_2, \ldots, a_n\}$: $i$th smallest elements:
  – Minimum and maximum.
  – Median, lower median, upper median

• Selection in expected/average linear time
  – Worst case running time
  – Prune-and-search

• Selection in worst-case linear time: