Computer Science & Engineering 423/823
Design and Analysis of Algorithms
Lecture 01 — Medians and Order Statistics (Chapter 9)

Stephen Scott
(Adapted from Vinodchandran N. Variyam)

sscott@cse.unl.edu
Introduction

- Given an array $A$ of $n$ distinct numbers, the $i$th order statistic of $A$ is its $i$th smallest element
  - $i = 1 \Rightarrow$ minimum
  - $i = n \Rightarrow$ maximum
  - $i = \lfloor (n + 1)/2 \rfloor \Rightarrow$ (lower) median
- E.g. if $A = [8, 5, 3, 10, 4, 12, 6]$ then min = 3, max = 12, median = 6, 3rd order stat = 5
- **Problem**: Given array $A$ of $n$ elements and a number $i \in \{1, \ldots, n\}$, find the $i$th order statistic of $A$
- There is an obvious solution to this problem. What is it? What is its time complexity?
  - Can we do better? What if we only focus on $i = 1$ or $i = n$?
Minimum($A$)

```plaintext
1  small = A[1]
2  for i = 2 to n do
3   if small > A[i] then
4     small = A[i]
5  end
6  return small
```
Efficiency of Minimum($A$)

- Loop is executed $n - 1$ times, each with one comparison
  - $\Rightarrow$ Total $n - 1$ comparisons
- Can we do better?
- **Lower Bound:** Any algorithm finding minimum of $n$ elements will need at least $n - 1$ comparisons
  - Proof of this comes from fact that no element of $A$ can be considered for elimination as the minimum until it’s been compared at least once
Correctness of Minimum($A$)

- Observe that the algorithm always maintains the **invariant** that at the end of each loop iteration, $small$ holds the minimum of $A[1 \cdots i]$
  - Easily shown by induction
- Correctness follows by observing that $i == n$ before **return** statement
Simultaneous Minimum and Maximum

- Given array \( A \) with \( n \) elements, find both its minimum and maximum.
- What is the obvious algorithm? What is its (non-asymptotic) time complexity?
- Can we do better?
MinAndMax($A, n$)

1. $large = \max(A[1], A[2])$
2. $small = \min(A[1], A[2])$
3. for $i = 2$ to $\lfloor n/2 \rfloor$ do
   4. $large = \max(large, \max(A[2i - 1], A[2i]))$
   5. $small = \min(small, \min(A[2i - 1], A[2i]))$
4. end
6. if $n$ is odd then
7.   $large = \max(large, A[n])$
8.   $small = \min(small, A[n])$
9. return $(large, small)$
Explanation of MinAndMax

- Idea: For each pair of values examined in the loop, compare them directly.
- For each such pair, compare the smaller one to *small* and the larger one to *large*.
- Example: $A = [8, 5, 3, 10, 4, 12, 6]$
Efficiency of MinAndMax

- How many comparisons does MinAndMax make?
- Initialization on Lines 1 and 2 requires only one comparison
- Each iteration through the loop requires one comparison between $A[2i - 1]$ and $A[2i]$ and then one comparison to each of large and small, for a total of three
- Lines 8 and 9 require one comparison each
- Total is at most $1 + 3\left\lfloor \frac{n}{2} \right\rfloor - 1 + 2 \leq 3\left\lfloor \frac{n}{2} \right\rfloor$, which is better than $2n - 3$ for finding minimum and maximum separately
Selection of the \( i \)th Smallest Value

- Now to the general problem: Given \( A \) and \( i \), return the \( i \)th smallest value in \( A \)
- Obvious solution is sort and return \( i \)th element
- Time complexity is \( \Theta(n \log n) \)
- Can we do better?
Selection of the $i$th Smallest Value (2)

- New algorithm: Divide and conquer strategy
- Idea: Somehow discard a constant fraction of the current array after spending only linear time
  - If we do that, we’ll get a better time complexity
  - More on this later
- Which fraction do we discard?
Select($A, p, r, i$)

```plaintext
1  if $p == r$ then
2      return $A[p]$
3  $q = \text{Partition}(A, p, r)$ // Like Partition in Quicksort
4  $k = q - p + 1$ // Size of $A[p \cdots q]$
5  if $i == k$ then
6      return $A[q]$ // Pivot value is the answer
7  else if $i < k$ then
8      return Select($A, p, q - 1, i$) // Answer is in left subarray
9  else
10     return Select($A, q + 1, r, i - k$) // Answer is in right subarray
11
Returns $i$th smallest element from $A[p \cdots r]$
```
What is Select Doing?

- Like in Quicksort, Select first calls Partition, which chooses a **pivot element** \( q \), then reorders \( A \) to put all elements \( \leq A[q] \) to the left of \( A[q] \) and all elements \( > A[q] \) to the right of \( A[q] \)
- E.g. if \( A = [1, 7, 5, 4, 2, 8, 6, 3] \) and pivot element is 5, then result is \( A' = [1, 4, 2, 3, 5, 7, 8, 6] \)
- If \( A[q] \) is the element we seek, then return it
- If sought element is in left subarray, then recursively search it, and ignore right subarray
- If sought element is in right subarray, then recursively search it, and ignore left subarray
Partition($A, p, r$)

1. $x = \text{ChoosePivotElement}(A, p, r)$ // Returns index of pivot
2. exchange $A[x]$ with $A[r]$
3. $i = p - 1$
4. for $j = p$ to $r - 1$ do
5.     if $A[j] \leq A[r]$ then
6.         $i = i + 1$
7.     exchange $A[i]$ with $A[j]$
8. end
10. return $i + 1$

Chooses a pivot element and partitions $A[p \cdots r]$ around it
Compare each element $A[j]$ to $x \ (\equiv 4)$ and swap with $A[i]$ if $A[j] \leq x$
Choosing a Pivot Element

- Choice of pivot element is critical to low time complexity
- Why?
- What is the best choice of pivot element to partition $A[p \cdots r]$?
Choosing a Pivot Element (2)

- Want to pivot on an element that it as close as possible to being the median
- Of course, we don’t know what that is
- Will do **median of medians** approach to select pivot element
Given (sub)array $A$ of $n$ elements, partition $A$ into $m = \lfloor n/5 \rfloor$ groups of 5 elements each, and at most one other group with the remaining $n \mod 5$ elements

Make an array $A' = [x_1, x_2, \ldots, x_{n/5}]$, where $x_i$ is median of group $i$, found by sorting (in constant time) group $i$

Call \textbf{Select}(\(A', 1, \lceil n/5 \rceil, \lfloor (\lceil n/5 \rceil + 1)/2 \rfloor\)) and use the returned element as the pivot
Example

Split into teams, and work this example on the board: Find the 4th smallest element of $A = [4, 9, 12, 17, 6, 5, 21, 14, 8, 11, 13, 29, 3]$
Show results for each step of Select, Partition, and ChoosePivotElement
Time Complexity

- Key to time complexity analysis is lower bounding the fraction of elements discarded at each recursive call to Select.
- On next slide, medians and median \((x)\) of medians are marked, arrows indicate what is guaranteed to be greater than what.
- Since \(x\) is less than at least half of the other medians (ignoring group with \(< 5\) elements and \(x\)’s group) and each of those medians is less than 2 elements, we get that the number of elements \(x\) is less than is at least

\[
3 \left( \left\lfloor \frac{1}{2} \left\lfloor \frac{n}{5} \right\rfloor \right\rfloor - 2 \right) \geq \frac{3n}{10} - 6 \geq n/4 \quad \text{(if } n \geq 120)\]

- Similar argument shows that at least \(3n/10 - 6 \geq n/4\) elements are less than \(x\).
- Thus, if \(n \geq 120\), each recursive call to Select is on at most \(3n/4\) elements.
Time Complexity (2)
Time Complexity (3)

- Now can develop a recurrence describing Select’s time complexity
- Let $T(n)$ represent total time for Select to run on input of size $n$
- Choosing a pivot element takes time $O(n)$ to split into size-5 groups and time $T(n/5)$ to recursively find the median of medians
- Once pivot element chosen, partitioning $n$ elements takes $O(n)$ time
- Recursive call to Select takes time at most $T(3n/4)$
- Thus we get
  \[ T(n) \leq T(n/5) + T(3n/4) + O(n) \]
- Can express as $T(\alpha n) + T(\beta n) + O(n)$ for $\alpha = 1/5$ and $\beta = 3/4$
- **Theorem:** For recurrences of the form $T(\alpha n) + T(\beta n) + O(n)$ for $\alpha + \beta < 1$, $T(n) = O(n)$
- Thus Select has time complexity $O(n)$
Proof of Theorem

Top $T(n)$ takes $O(n)$ time ($= cn$ for some constant $c$). Then calls to $T(\alpha n)$ and $T(\beta n)$, which take a total of $(\alpha + \beta)cn$ time, and so on.

Summing these infinitely yields (since $\alpha + \beta < 1$)

$$cn(1 + (\alpha + \beta) + (\alpha + \beta)^2 + \cdots) = \frac{cn}{1 - (\alpha + \beta)} = c'n = O(n)$$
Master Method

- Another useful tool for analyzing recurrences

**Theorem:** Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined as $T(n) = aT(n/b) + f(n)$. Then $T(n)$ is bounded as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for constant $c < 1$ and sufficiently large $n$, then $T(n) = \Theta(f(n))$

- E.g. for Select, can apply theorem on $T(n) < 2T(3n/4) + O(n)$ (note the slack introduced) with $a = 2$, $b = 4/3$, $\epsilon = 1.4$ and get

$$T(n) = O\left(n^{\log_{4/3} 2}\right) = O\left(n^{2.41}\right)$$

⇒ Not as tight for this recurrence