Introduction to Algorithms
6.046J/18.401J

Lecture 5
Prof. Manolis Kellis

Today

• Order statistics (e.g., finding median)
• Two \(O(n)\) time algorithms:
  – Randomized: similar to Quicksort
  – Deterministic: quite tricky
• Both are examples of divide and conquer

Finding \(i^{th}\) smallest element

Select the \(i^{th}\) smallest of \(n\) elements (the element with \(\text{rank } i\)).

- \(i = 1\): minimum;
- \(i = n\): maximum;
- \(i = \left\lfloor \frac{n+1}{2} \right\rfloor \) or \(\left\lceil \frac{n+1}{2} \right\rceil\): median.

Example: finding the median \(\left\lfloor \frac{n+1}{2} \right\rceil\)-order statistic. Robust to outlier effects, picking an actual element

Ex: Find 7th smallest element

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
\end{array}
\]

\(i = 7\)

- How did you do it?
  – Find 2, 3, 5, 6, 8, in turn \(\Rightarrow O(n^2)\)
  – Find \(n-i=2\)nd largest \(\Rightarrow O(n)\) in this example
    – But: \(O(n^2)\) in general case
- In the general case:
  – Finding min/max: \(O(n)\) (special case, 0th nth order).
  – Finding \(i^{th}\) element by repeated search: \(O(n^2)\).
  – Finding \(i^{th}\) statistic by sorting: \(O(n \log n)\).
- Today:
  – How to do it in \(O(n)\) time average case, randomized
  – How to do it in \(O(n)\) time worst-case, deterministic

Randomized Algorithm for Finding the \(i^{th}\) element

• Divide and Conquer Approach
• Main idea: PARTITION

\[
\begin{array}{ccc}
\leq x & x & \geq x \\
k & p & q
\end{array}
\]

- If \(i < k\), recurse on the left
- If \(i > k\), recurse on the right
- Otherwise, output \(x\)

Randomized Divide-and-Conquer

\[
\text{RAND-SELECT}(A, p, r, i)
\]

\[
\begin{array}{l}
\text{if } p = r \text{ then return } A[p] \\
q \leftarrow \text{RAND-PARTITION}(A, p, r) \\
k \leftarrow q - p + 1 \\
\text{if } i = k \text{ then return } A[q] \\
\text{if } i < k \text{ then return RAND-SELECT}(A, q - 1, i) \\
\text{else return RAND-SELECT}(A, q + 1, r, i - k)
\end{array}
\]

\[
\begin{array}{ccc}
p & q & r
\end{array}
\]
Partitioning subroutine

\[ \text{PARTITION}(A, p, r) \]

\[ x \leftarrow A[p] \]
\[ \text{pivot} = A[p] \]
\[ i \leftarrow p \]

for \( j \leftarrow p + 1 \) to \( r \)

\[ \text{if } A[j] \leq x \]

\[ i \leftarrow i + 1 \]

exchange \( A[i] \leftrightarrow A[j] \)

return \( i \)

Inv
v

\[ \begin{array}{c|c|c|c|c}
   p & i & j & r & ? \\
   \hline
   x & \leq x & \geq x & ? & ?
   \end{array} \]

Example

Select the \( i = 7 \)th smallest:

\[ \begin{array}{cccccccc}
   6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
   \hline
   \end{array} \]

\( i = 7 \)

pivot

Partition:

\[ \begin{array}{cccccccc}
   2 & 5 & 3 & 6 & 8 & 13 & 10 & 11 \\
   \hline
   \end{array} \]

\( k = 4 \)

Select the \( 7 - 4 = 3 \)rd smallest recursively.

Analysis #1

• What is the worst-case running time?
  \[ T(n) = T(n-1) + \Theta(n) \]
  \( = \Theta(n^2) \)

Unlucky:

Recall that a lucky partition splits into arrays with size ratio at most 9:1

What if all partitions are lucky?

Lucky:

\[ T(n) = T(9n/10) + \Theta(n) \]
\[ = \Theta(n) \]

Case 3

Expected Running Time

• The probability that a random pivot induces lucky partition is at least 8/10 (Lecture 4)
  \[ t_i \]
  \( = \Omega(n) \)

• The total time is at most...

\[ T = t_1 n + t_2 (9/10) n + t_3 (9/10)^2 n + \ldots \]

• The total expected time is at most:

\[ E[T] = E[t_1 n + E[t_2] (9/10) n + E[t_3] (9/10)^2 n + \ldots] \]
\[ = 10/8 * [n + (9/10)n + \ldots] \]
\[ = O(n) \]

Digression: 9 to 1

• Do we need to define the lucky partition as 9:1 balanced?

No. Suffices to say that both sides have size \( \geq \alpha n \), for \( \alpha < \frac{1}{2} \)

• Then probability of getting a lucky partition is

\[ 1 - 2\alpha \]

Analysis #2: Random variables!

The analysis follows that of randomized quicksort, but it’s a little different.

Let \( T(n) \) = the random variable for the running time of RAND-SELECT on an input of size \( n \), assuming random numbers are independent.

For \( k = 0, 1, \ldots, n-1 \), define the indicator random variable

\[ X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases} \]
Analysis (continued)

To obtain an upper bound, assume that the i-th element always falls in the larger side of the partition:

\[
T(n) = \begin{cases} 
T(\max\{0, n-1\}) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\
T(\max\{1, n-2\}) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\
T(\max\{n-1, 0\}) + \Theta(n) & \text{if } n-1 : 0 \text{ split,}
\end{cases}
\]

= \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)),

Calculating expectation

\[
E[T(n)] = \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))]
\]

= \sum_{k=0}^{n-1} E[X_k] E[T(\max\{k, n-k-1\}) + \Theta(n)]

= \sum_{k=0}^{n-1} E[X_k] E[T(\max\{k, n-k-1\})] + \sum_{k=0}^{n-1} E[X_k] \Theta(n)

= \sum_{k=0}^{n-1} E[X_k] \frac{\Theta(n)}{n}

= \frac{\Theta(n)}{n} \sum_{k=0}^{n-1} E[X_k]

= \frac{\Theta(n)}{n} \cdot \frac{n}{n}

= \Theta(n)

\[
E[X_k] = \frac{1}{n}
\]

Calculating expectation

\[
E[T(n)] = \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n))
\]

Take expectations of both sides.

Calculating expectation

\[
E[T(n)] = \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))]
\]

= \sum_{k=0}^{n-1} E[X_k] E[T(\max\{k, n-k-1\}) + \Theta(n)]

= \sum_{k=0}^{n-1} E[X_k] E[T(\max\{k, n-k-1\})] + \sum_{k=0}^{n-1} E[X_k] \Theta(n)

Independence of \( X_k \) from other random choices.
Calculating expectation

\[ E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right] \]
\[ = \sum_{k=0}^{n-1} E[X_k T(\max\{k, n-k-1\}) + \Theta(n)) \]
\[ = \sum_{k=0}^{n-1} E[X_k] E[T(\max\{k, n-k-1\}) + \Theta(n)] \]
\[ = \sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\}) + \Theta(n)] \]
\[ = \sum_{k=0}^{n-1} E[T(k)] + \Theta(n) \]
\[ \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + \Theta(n) \]
Upper terms appear twice.

Hairy recurrence

(But not quite as hairy as the quicksort one.)

\[ E[T(n)] = 2 \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + \Theta(n) \]

Prove: \( E[T(n)] \leq cn \) for constant \( c > 0 \).

- The constant \( c \) can be chosen large enough so that \( E[T(n)] \leq cn \) for the base cases.

Use fact: \( \sum_{k=\lceil n/2 \rceil}^{n-1} k \leq \frac{3}{8} n^2 \) (exercise).

Substitution method

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

Substitute inductive hypothesis.

Calculating expectation

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} c k + \Theta(n) \]
\[ \leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \]

Use fact.

Substitution method

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} c k + \Theta(n) \]
\[ \leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \]
\[ = cn - \left( \frac{cn}{4} - \Theta(n) \right) \]

Express as desired – residual.
Substitution method

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} c k + \Theta(n) \]
\[ \leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \]
\[ = cn - \left( \frac{cn}{4} - \Theta(n) \right) \]
\[ \leq cn, \]

if \( c \) is chosen large enough so that \( cn/4 \) dominates the \( \Theta(n) \).

Summary of randomized order-statistic selection

- Works fast: expected time \( \Theta(n) \).
- Excellent algorithm in practice.
- But, the worst case is very bad: \( \Theta(n^2) \).

Deterministic linear-time order-statistic selection

- Works fast: expected time \( \Theta(n) \).
- Excellent algorithm in practice.
- But, the worst case is very bad: \( \Theta(n^2) \).

Q. Can we achieve: worst case \( O(n) \)?
A. Yes, due to [Blum-Floyd-Pratt-Rivest-Tarjan ‘73].

IDEA: Generate a good pivot recursively.

Worst-case linear-time order statistics

\textbf{SELECT} \((i, n)\)

1. Divide the \( n \) elements into groups of 5. Find the median of each 5-element group by hand.
2. Recursively \textbf{SELECT} the median \( x \) of the \( \lceil n/5 \rceil \) group medians to be the pivot.
3. Partition around the pivot \( x \). Let \( k = \text{rank}(x) \).
4. if \( i = k \) then \textbf{return} \( x \)
   
   else if \( i < k \)
   
   then recursively \textbf{SELECT} the \( i \)th smallest element in the lower part
   
   else recursively \textbf{SELECT} the \((i-k)\)th smallest element in the upper part

Choosing the pivot

1. Divide the \( n \) elements into groups of 5.
Choosing the pivot

1. Divide the $n$ elements into groups of 5. Find the median of each 5-element group by rote (insertion sort will do).

2. Recursively select the median $x$ of the $\lceil n/5 \rceil$ group medians to be the pivot.

Analysis

At least half the group medians are $\leq x$, which is at least $\lceil \lceil n/5 \rceil/2 \rceil = \lceil n/10 \rceil$ group medians.

• Therefore, at least $3 \lceil n/10 \rceil$ elements are $\leq x$.

• Similarly, at least $3 \lceil n/10 \rceil$ elements are $\geq x$.

Developing the recurrence

\[
\begin{align*}
T(n) & \quad \text{SELECT}(i, n) \\
\Theta(n) & \quad 1. \text{Divide the } n \text{ elements into groups of } 5. \text{ Find the median of each } 5\text{-element group by rote.} \\
T(n/5) & \quad 2. \text{Recursively SELECT the median } x \text{ of the } \lceil n/5 \rceil \text{ group medians to be the pivot.} \\
\Theta(n) & \quad 3. \text{Partition around the pivot } x. \text{ Let } k = \text{rank}(x). \\
T(7n/10) & \quad 4. \begin{cases} 
\text{if } i = k \text{ then return } x \\
\text{else } i < k \\
\text{then recursively SELECT the } i\text{th smallest element in the lower part} \\
\text{else recursively SELECT the } (i-k)\text{th smallest element in the upper part}
\end{cases}
\end{align*}
\]
Solving the recurrence

\[ T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n) \]

Substitution method:

\[ T(n) \leq \frac{1}{5}cn + \frac{7}{10}cn + \Theta(n) \]
\[ = \frac{18}{20}cn + \Theta(n) \]
\[ = cn - \left(\frac{2}{20}cn - \Theta(n)\right) \]
\[ \leq cn \]

if \( c \) is chosen large enough to handle the \( \Theta(n) \).

Minor simplification

- For \( n \geq 50 \), we have \( \lfloor n/10 \rfloor \geq n/4 \).
- Therefore, for \( n \geq 50 \) the recursive call to \( \text{SELECT} \) in Step 4 is executed recursively on \( \leq 3n/4 \) elements.
- Thus, the recurrence for running time can assume that Step 4 takes time \( T(3n/4) \) in the worst case.
- For \( n < 50 \), we know that the worst-case time is \( T(n) = \Theta(1) \).

Conclusions

- Since the work at each level of recursion is a constant fraction \((18/20)\) smaller, the work per level is a geometric series dominated by the linear work at the root.
- In practice, this algorithm runs slowly, because the constant in front of \( n \) is large.
- The randomized algorithm is far more practical.

Exercise: Why not divide into groups of 3?