CS 341: ALGORITHMS

Lecture 4: divide & conquer III
Readings: see website

Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
THE SELECTION PROBLEM

NATURAL SELECTION

in progress...
THE SELECTION PROBLEM

• Input: An array $A$ containing $n$ distinct integer values, and an integer $k$ between 1 and $n$

• Output: The $k$-th smallest integer in $A$

• Minimum is a special case where $k = 1$

• Median is a special case where $k = \frac{n}{2}$

• Maximum is a special case where $k = n$

• Simple algorithm for solving selection?
Suppose we choose a **pivot** element $y$ in the array $A$, and we **restructure** $A$ so that all elements less than $y$ precede $y$ in $A$, and all elements greater than $y$ occur after $y$ in $A$. (This is exactly what is done in **Quicksort**, and it takes **linear time**.)

$$\text{Restructure}(A, y)$$

Number of elements on each side depend on the **value** $y$…
What's the $k$-th smallest element of $A$?

- If $k = i_y$ then $y$
- If $k < i_y$ then the $k$th smallest in $A_L$
- If $k > i_y$ then the $(k - i_y)$th smallest in $A_R$
QuickSelect(k, A[1..n])
    if n = 1 then return A[1]  // base case
    y = A[1]                      // pick an arbitrary pivot
    (AL, AR, iy) = Restructure(A, y)
    if k == iy return y
    else if k < iy then return QuickSelect(k, AL)
    else /* k > iy */ return QuickSelect(k - iy, AR)

Restructure(A[1..n], y)
    AL = new array[1..n]          // allocate more than enough
    AR = new array[1..n]          // to avoid need for expansion
    nL = 0, nR = 0
    for i = 1 .. n
        if A[i] < y then AL[nL++] = A[i]
        else A[i] > y then AR[nR++] = A[i]
    return (AL, AR, nL+1)          // nL+1 is the new index of y
OVERLY OPTIMISTIC ANALYSIS 😊

If \( i_y = \frac{n}{2} \), then we recurse on \( \sim \frac{n}{2} \) elements.

If we could always recurse on \( \frac{n}{2} \) elements then

- We would get \( T(n) = T\left(\frac{n}{2}\right) + \Theta(n) \)
- Which would yield \( a = 1, b = 2, y = 1, x = \log_2 1 = 0, y > x \) and \( T(n) \in \Theta(n^y) = \Theta(n) \) by the Master theorem.

But we don't always recurse on \( \frac{n}{2} \) elements!
WORST-CASE ANALYSIS

If we always get $i_y = 1$ and recurse on the right, then

- We get $T(n) = T(n - 1) + \Theta(n)$
- By the substitution method this is $\Theta(n^2)$

So, sometimes the pivot is good, sometimes it’s bad…

What about the average case?
AVERAGE-CASE ANALYSIS

• Definition: we say a pivot $y$ is good if $i_y \in \left( \frac{n}{4}, \frac{3n}{4} \right)$

• For any good pivot, we recurse on at most $\frac{3n}{4}$ elements

• Probability of an arbitrary pivot being good? $p = \frac{1}{2}$
PROOF SKETCH

• Since probability of a good pivot is $\frac{1}{2}$,
• on average, every two recursive calls, we will encounter a good pivot
• Encountering a good pivot reduces problem size to at most $\frac{3n}{4}$
• So, problem size is reduced to $\frac{3n}{4}$ after expected linear work

• Average case recurrence: $T(n) = T\left(\frac{3n}{4}\right) + \Theta(n)$
  • $T(n) \in \Theta(n)$
Here is a more rigorous proof of the average-case complexity: We say the algorithm is in phase $j$ if the current subarray has size $s$, where

$$n \left( \frac{3}{4} \right)^{j+1} < s \leq n \left( \frac{3}{4} \right)^j.$$

Let $X_j$ be a random variable that denotes the amount of computation time occurring in phase $j$. If the pivot is in the middle half of the current subarray, then we transition from phase $j$ to phase $j + 1$. This occurs with probability $1/2$, so the expected number of recursive calls in phase $j$ is 2. The computing time for each recursive call is linear in the size of the current subarray, so $E[X_j] \leq 2cn(3/4)^j$ (where $E[\cdot]$ denotes the expectation of a random variable). The total time of the algorithm is given by $X = \sum_{j \geq 0} X_j$. Therefore

$$E[X] = \sum_{j \geq 0} E[X_j] \leq 2cn \sum_{j \geq 0} (3/4)^j = 8cn \in O(n).$$

This is just for your notes, in case you want to know how you'd do this analysis formally.
TAKING SELECTION FURTHER

• We just showed:
  • QuickSelect with average case runtime in $O(n)$
• Next up:
  • Median-of-medians QuickSelect (MOMQuickSelect)
  • worst case runtime in $O(n)$

Relies on getting a good pivot within $O(1)$ recursive calls on average

Must get a good pivot within $O(1)$ recursive calls always

The algorithm we will see picks a good pivot in every recursive call
HIGH LEVEL ALGORITHM

• Similar to QuickSelect
  • **Choose** a pivot
  • Move smaller elements to the left of the pivot, and larger elements to the right of the pivot
  • Recursively call MOMQuickSelect on one subarray
  • Only difference is **how** we choose the pivot
  • **Always** want to pick a **good pivot**
### ALWAYS PICKING A GOOD PIVOT

Example input $A[1...50]$: 11, 38, 6, 21, 20, 17, 14, 9, 7, 5, 8, 34, 49, 47, 28, 18, 44, 31, 46, 48, 27, 4, 2, 50, 23, 45, 3, 13, 43, 22, 10, 32, 35, 41, 24, 1, 30, 12, 15, 26, 16, 19, 36, 33, 37, 39, 25, 40, 29, 42

<table>
<thead>
<tr>
<th>Group into rows of 5</th>
<th>Find median of each row</th>
<th>Build array of medians</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 38 6 21 20 17 14 9 7 5 8 34 49 47 28 18 44 31 46 48 27 4 2 50 23 45 3 13 43 22 10 32 35 41 24 1 30 12 15 26 16 19 36 33 37 39 25 40 29 42</td>
<td>11 38 6 21 20 17 14 9 7 5 8 34 49 47 28 18 44 31 46 48 27 4 2 50 23 45 3 13 43 22 10 32 35 41 24 1 30 12 15 26 16 19 36 33 37 39 25 40 29 42</td>
<td>20, 9, 34, 44, 23, 22, 32, 15, 33, 39</td>
</tr>
</tbody>
</table>

**Time complexity** for this step?

**Time complexity** for this step?

Recursively find the median of these medians: 23

Recursive problem size?
### How Good Is the Pivot 23?

#### Recall: Median of Each Row

<table>
<thead>
<tr>
<th>11</th>
<th>38</th>
<th>6</th>
<th>21</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>34</td>
<td>49</td>
<td>47</td>
<td>28</td>
</tr>
<tr>
<td>18</td>
<td>44</td>
<td>31</td>
<td>46</td>
<td>48</td>
</tr>
<tr>
<td>27</td>
<td>4</td>
<td>2</td>
<td>50</td>
<td>23</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
<td>13</td>
<td>43</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>35</td>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td>12</td>
<td>15</td>
<td>26</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>36</td>
<td>33</td>
<td>37</td>
</tr>
<tr>
<td>39</td>
<td>25</td>
<td>40</td>
<td>29</td>
<td>42</td>
</tr>
</tbody>
</table>

#### Imagine Sorting Each Row:

<table>
<thead>
<tr>
<th>6</th>
<th>11</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>34</td>
</tr>
<tr>
<td>18</td>
<td>31</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>33</td>
</tr>
<tr>
<td>25</td>
<td>29</td>
<td>39</td>
</tr>
</tbody>
</table>

#### Then Ordering Rows by Medians:

<table>
<thead>
<tr>
<th>5</th>
<th>7</th>
<th>9</th>
<th>14</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>15</td>
<td>26</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>20</td>
<td>21</td>
<td>38</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>22</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>23</td>
<td>27</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>32</td>
<td>35</td>
<td>41</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>33</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>34</td>
<td>47</td>
<td>49</td>
</tr>
<tr>
<td>39</td>
<td>40</td>
<td>42</td>
<td>46</td>
<td>48</td>
</tr>
<tr>
<td>18</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- # elements ≤ 23 is at least 3(5). This is at least 3/10ths of our 50-element input, or $\frac{3n}{10}$.
- # elements ≥ 23 is at least 3(6). This is > 3/10ths of our 50-element input.

So, after restructuring, pivot 23 must have at least $\frac{3n}{10}$ elements before and after it.

We recurse on $A_L$ or $A_R$, and both have size at most $\frac{7n}{10}$.

This is a good pivot!
MOMQuickSelect(k, n, A)
   // base case
   if n <= 14 then sort(A) and return A[k]

   // divide and conquer to find medians
   r = (n-5) / 10
   medians[1..(2*r+1)] = new array
   for i = 1..(2*r+1)
      B[1..5] = A[(5*(i-1)+1)..(5*i)]
      sort(B)
      medians[i] = B[3]

   y = MOMQuickSelect(r+1, 2*r+1, medians)

   // divide and conquer to find rank k
   (AL, AR, iy) = Restructure(A, y)
   if k == iy then return y
   else if k < iy then return MOMQuickSelect(k, iy-1, AL)
   else /* k > iy */ then return MOMQuickSelect(k-iy, n-iy, AR)
MOMQuickSelect(k, n, A)

// base case
if n <= 14 then sort(A) and return A[k]

// divide and conquer to find medians
r = (n-5) / 10
medians[1..(2*r+1)] = new array
for i = 1..(2*r+1)
    B[1..5] = A[(5*(i-1)+1)..(5*i)]
sort(B)
    medians[i] = B[3]

y = MOMQuickSelect(r+1, 2*r+1, medians)

// divide and conquer to find rank k
(AL, AR, iy) = Restructure(A, y)
if k == iy then return y
else if k < iy then return MOMQuickSelect(k, iy-1, AL)
else /* k > iy */ then return MOMQuickSelect(k-iy, n-iy, AR)
MOMQuickSelect\( (k = 11, n = 21, A) \)

11, 38, 6, 21, 20, 17, 14, 9, 7, 5, 8, 34, 49, 47, 28, 18, 44, 31, 46, 48, 27

// base case
if \( n \leq 14 \) then sort\( (A) \) and return \( A[k] \)

// divide and conquer to find medians
\( r = (n-5) / 10 \)
\( \text{medians}[1..(2\times r+1)] \) = new array
for \( i = 1..(2\times r+1) \)
\( B[1..5] = A[(5\times(i-1)+1)\ldots(5\times i)] \)
sort\( (B) \)
\( \text{medians}[i] = B[3] \)

\( y = \text{MOMQuickSelect}(r+1, 2\times r+1, \text{medians}) \)

// divide and conquer to find rank \( k \)
\( (AL, AR, iy) = \text{Restructure}(A, y) \)
if \( k == iy \) then return \( y \)
else if \( k < iy \) then return \( \text{MOMQuickSelect}(k, iy-1, AL) \)
else /* \( k > iy */ then return \( \text{MOMQuickSelect}(k-iy, n-iy, AR) \)

**Restructure** \( (A, y = 20) \) ⇒

\( A_L = [11, 6, 17, 14, 9, 7, 5, 8, 18] \)
\( A_R = [38, 21, 34, 49, 47, 28, 44, 31, 46, 48, 27] \)
\( iy = |A_L| + 1 = 10 \)

\( k = 11 \) \( > \) \( iy = 10 \)

\( k - iy = 1 \) \( n - iy = 10 \)

**MOMQuickSelect** \( (1, 10, A_R) \) ⇒ 21
// base case
if n <= 14 then sort(A) and return A[k]

// divide and conquer to find medians
r = (n-5) / 10
medians[1..(2*r+1)] = new array
for i = 1..(2*r+1)
    B[1..5] = A[(5*(i-1)+1)..(5*i)]
    sort(B)
    medians[i] = B[3]
y = MOMQuickSelect(r+1, 2*r+1, medians)

// divide and conquer to find rank k
(AL, AR, iy) = Restructure(A, y)
if k == iy then return y
else if k < iy then return MOMQuickSelect(k, iy-1, AL)
else /* k > iy */ then return MOMQuickSelect(k-iy, n-iy, AR)

3(r + 1) elements ≤ y
3(r + 1) elements ≥ y
So problem size shrinks by at least 3(r + 1)
Observe n = 10*r + 5
How much does the problem shrink?

- Shrinks by at least $3(r + 1)$
- Problem size $\approx n = 10r + 5$
- Subproblem size $\leq n - Shrink = n - 3(r + 1)$
  - $= 10r + 5 - 3r - 3 = 7r + 2$
- Express in terms of $n$ using $r = \left\lfloor \frac{n-5}{10} \right\rfloor$
  - Subproblem size $\leq 7\left\lfloor \frac{n-5}{10} \right\rfloor + 2 \leq 7 \frac{n-5}{10} + 2$
  - $= \frac{7n}{10} - 7 \left( \frac{5}{10} \right) + 2 = \frac{7n}{10} - \frac{3}{2} \leq \frac{7n}{10}$
// base case
if n <= 14 then sort(A) and return A[k]

// divide and conquer to find medians
r = (n-5) / 10
medians[1..(2*r+1)] = new array
for i = 1..(2*r+1)
    B[1..5] = A[(5*(i-1)+1)..(5*i)]
sort(B)
medians[i] = B[3]
y = MOMQuickSelect(r+1, 2*r+1, medians)

// divide and conquer to find rank k
(AL, AR, iy) = Restructure(A, y)
if k == iy then return y
else if k < iy then return MOMQuickSelect(k, iy-1, AL)
else /* k > iy */ then return MOMQuickSelect(k-iy, n-iy, AR)

T(n) ∈ O(n) + T(n/5) + T(7n/10) if n ≥ 15
T(n) ∈ O(1) if n ≤ 14
The key fact is that $1/5 + 7/10 = 9/10 < 1$.

The fact that $T(n) \in \Theta(n)$ can be proven formally using guess-and-check (induction) or informally using the recursion tree method.

\[
T(n) \in O(n) + T(n/5) + T(7n/10) \quad \text{if } n \geq 15
\]
\[
T(n) \in O(1) \quad \text{if } n \leq 14
\]

\[
\sum_{i=0}^{8} n \left(\frac{9}{10}\right)^i = 10n \in \Theta(n)
\]
Let \( T(n) = c'n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \) where \( c' > 0 \)

Want to prove: \( T(n) = cn \) for some \( c > 0 \)

Note \( c \) and \( c' \) are independent constants
  - \( c' \) comes from the work at each level of recursion being \( O(n) \)
  - \( c \) is a positive constant we are trying to show exists

I.H.: Suppose \( \exists c > 0 : T(n') = cn' \) for \( 15 \leq n' < n \)

\( T(n) = c'n + c \frac{n}{5} + c \frac{7n}{10} \) (by inductive hypoth.)

\( T(n) = cn \) (want this to be true)

\( \iff c'n + c \frac{n}{5} + c \frac{7n}{10} = cn \) (equivalently)

\( \iff c' + c \frac{1}{5} + c \frac{7}{10} = c \iff c = 10c' \) (by algebra)