Quickselect

Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Administrivia

- Assignment 2 is out
  - First part due on Friday by 4:30pm
  - Second part due next Friday by 4:30pm
  - Demos in the lab

- Quiz 2 on Thursday
  - Coverage through today
    (topics include running time, sorting)
  - Closed book / closed note
Recap from last time

- We can solve the selection problem by sorting the numbers first

- We’ve learned two ways to do this so far:
  1. Selection sort
  2. Quicksort

Quicksort

1. Pick an element (pivot)
2. **Partition** the array into elements < pivot, = to pivot, and > pivot
3. Quicksort these smaller arrays separately

- What is the worst-case running time?
- What is the expected running time (on a random input)?
Back to the selection problem

- Can solve with quicksort
  - Faster (on average) than “repeated remove biggest”
- Is there a better way?

- Rev. Charles L. Dodgson’s problem
  - Based on how to run a tennis tournament
  - Specifically, how to award 2\textsuperscript{nd} prize fairly

- How many teams were in the tournament?
- How many games were played?
- Which is the second-best team?
Standard Tournament

- Example
  
  \[
  [ 8 \ 3 \ 1 \ 2 \ 4 \ 6 \ 7 \ 5 ]
  \]

- Compare everyone to their neighbor, keep the larger one

  \[
  [ 8 \ 2 \ 6 \ 7 ]
  [ 8 \ 7 ]
  [ 8 ]
  \]

Finding the second best team

- Could use quicksort to sort the teams

- Step 1: Choose one team as a pivot (say, Arizona)
- Step 2: Arizona plays every team
- Step 3: Put all teams worse than Arizona in Group 1, all teams better than Arizona in Group 2 (no ties allowed)
- Step 4: Recurse on Groups 1 and 2
- ... eventually will rank all the teams ...
Quicksort Tournament

- (Note this is a bit silly – AZ plays 63 games)
- This gives us a ranking of all teams
- What if we just care about finding the 2nd-best team?

Modifying quicksort to select

- Suppose Arizona beats 36 teams, and loses to 27 teams

If we just want to know the 2nd-best team, how can we save time?
Modifying quicksort to select – Finding the 2\textsuperscript{nd} best team

36 teams < \[
\begin{array}{c}
\text{A} \\
\text{Group 1}
\end{array}
\] < 27 teams

< 27 teams

\[
\begin{array}{c}
\text{Group 2}
\end{array}
\]

16 teams < \[
\begin{array}{c}
\text{Cornell} \\
\text{Group 2.1}
\end{array}
\] < 10 teams

< 10 teams

\[
\begin{array}{c}
\text{Group 2.2}
\end{array}
\]

7 teams < \[
\begin{array}{c}
\text{\textbullet} \\
\text{Group 1.2}
\end{array}
\] < 2 teams

Q: Which group do we visit next?

The 32\textsuperscript{nd} best team overall is the 4\textsuperscript{th} best team in Group 1

---

Modifying quicksort to select – Finding the 32\textsuperscript{nd} best team

36 teams < \[
\begin{array}{c}
\text{A} \\
\text{Group 1}
\end{array}
\] < 27 teams

< 27 teams

\[
\begin{array}{c}
\text{Group 2}
\end{array}
\]

20 teams < \[
\begin{array}{c}
\text{George Mason University} \\
\text{Group 1.1}
\end{array}
\] < 15 teams

< 15 teams

\[
\begin{array}{c}
\text{Group 1.2}
\end{array}
\]

Q: Which group do we visit next?

The 32\textsuperscript{nd} best team overall is the 4\textsuperscript{th} best team in Group 1
Find $k^{th}$ largest element in $A$ (< than $k-1$ others)

$A = [6.0, 5.4, 5.5, 6.2, 5.3, 5.0, 5.9]$

**MODIFIED QUICKSORT**($A, k$):

- Pick an element in $A$ as the pivot, call it $x$
- Divide $A$ into $A_1$ (<$x$), $A_2$ ($=x$), $A_3$ (>=$x$)
- If $k < \text{length}(A_3)$
  - MODIFIED QUICKSORT ($A_3, k$)
- If $k > \text{length}(A_2) + \text{length}(A_3)$
  - Let $j = k - [\text{length}(A_2) + \text{length}(A_3)]$
  - MODIFIED QUICKSORT ($A_1, j$)
- Otherwise, return $x$

**Modified quicksort**

**MODIFIED QUICKSORT**($A, k$):

- Pick an element in $A$ as the pivot, call it $x$
- Divide $A$ into $A_1$ (<$x$), $A_2$ ($=x$), $A_3$ (>=$x$)
- If $k < \text{length}(A_3)$
  - Find the element <$k$ others in $A_3$
- If $k > \text{length}(A_2) + \text{length}(A_3)$
  - Let $j = k - [\text{length}(A_2) + \text{length}(A_3)]$
  - Find the element <$j$ others in $A_1$
- Otherwise, return $x$

- We’ll call this *quickselect*
- Let’s consider the running time...
What is the running time of:

- Finding the 1st element?
  - $O(1)$ (effort doesn’t depend on input)

- Finding the biggest element?
  - $O(n)$ (constant work per input element)

- Finding the median by repeatedly finding and removing the biggest element?
  - $O(n^2)$ (linear work per input element)

- Finding the median using quickselect?
  - Worst case? $O(\_\_\_\_\_\_\_)$
  - Best case? $O(\_\_\_\_\_\_\_)$

Quickselect – “medium” case

- Suppose we split the array in half each time (i.e., happen to choose the median as the pivot)

- How many comparisons will there be?
How many comparisons? ("medium" case)

- Suppose \( \text{length}(A) == n \)
- Round 1: Compare \( n \) elements to the pivot
  ... now break the array in half, quickselect one half ...
- Round 2: For remaining half, compare \( n / 2 \) elements to the pivot (total # comparisons = \( n / 2 \))
  ... now break the half in half ...
- Round 3: For remaining quarter, compare \( n / 4 \) elements to the pivot (total # comparisons = \( n / 4 \))

Number of comparisons = 
\[
n + n / 2 + n / 4 + n / 8 + \ldots + 1
\]

= ?

→ The "medium" case is \( O(n) \)!
Quickselect

- For random input this method actually runs in linear time (beyond the scope of this class)
- The worst case is still bad
- Quickselect gives us a way to find the $k^{th}$ element without actually sorting the array!

Quickselect

- It’s possible to select in *guaranteed* linear time (1973)
  - Rev. Dodgson’s problem
  - But the code is a little messy
    • And the analysis is messier
- Beyond the scope of this course
Questions?

Back to the lightstick

- By using quickselect we can find the 5% largest (or smallest) element
  - This allows us to efficiently compute the trimmed mean
What about the median?

- Another way to avoid our bad data points:
  - Use the median instead of the mean

Median vector

- Mean, like median, was defined in 1D
  - For a 2D mean we used the centroid
  - Mean of x coordinates and y coordinates separately
    - Call this the “mean vector”

- Does this work for the median also?
What is the median vector?

- In 1900, statisticians wanted to find the “geographical center of the population” to quantify westward shift.
- Why not the centroid?
  - Someone being born in San Francisco changes the centroid much more than someone being born in Indiana.
- What about the “median vector”?
  - Take the median of the $x$ coordinates and the median of the $y$ coordinates separately.
Median vector

- A little thought will show you that this doesn’t really make a lot of sense
  - Nonetheless, it’s a common solution, and we will implement it for CS1114
  - In situations like ours it works pretty well
- It’s almost never an actual datapoint
- It depends upon rotations!

Can we do even better?

- None of what we described works that well if we have widely scattered red pixels
  - And we can’t figure out lightstick orientation
- Is it possible to do even better?
  - Yes!
- We will focus on:
  - Finding “blobs” (connected red pixels)
  - Summarizing the shape of a blob
  - Computing orientation from this
- We’ll need brand new tricks!
• The lightstick forms a large “blob” in the thresholded image (among other blobs)

**What is a blob?**

```
1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 0 0 0 0 0
```
Finding blobs

1. Pick a 1 to start with, where you don’t know which blob it is in
   - When there aren’t any, you’re done
2. Give it a new blob color
3. Assign the same blob color to each pixel that is part of the same blob
## Finding blobs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

## Finding blobs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Finding blobs

\[
\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Finding blobs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>