Randomized Algorithms
Part One
Announcements

• Problem Set 2 due right now if you're using a late period.
  • Solutions released right after lecture.
• Julie's Tuesday office hours this week will be remote office hours. Details emailed out tomorrow.
Outline for Today

- **Randomized Algorithms**
  - How can randomness help solve problems?
- **Quickselect**
  - Can we do away with median-of-medians?
- **Techniques in Randomization**
  - Linearity of expectation, the union bound, and other tricks.
Randomized Algorithms
Deterministic Algorithms

- The algorithms we've seen so far have been deterministic.
- We want to aim for properties like
  - Good worst-case behavior.
  - Getting exact solutions.
- Much of our complexity arises from the fact that there is little flexibility here.
- Often find complex algorithms with nuanced correctness proofs.
Randomized Algorithms

- A **randomized algorithm** is an algorithm that incorporates randomness as part of its operation.

- Often aim for properties like
  - Good *average-case* behavior.
  - Getting exact answers *with high probability*.
  - Getting answers that are *close* to the right answer.

- Often find very simple algorithms with dense but clean analyses.
Where We're Going

Motivating examples:

- Quickselect and quicksort are **Las Vegas algorithms**: they always find the right answer, but might take a while to do so.
- Karger's algorithm is a **Monte Carlo algorithm**: it might not always find the right answer, but has dependable performance.
- Hash tables with universal hash functions are **randomized data structures** that have high performance due to randomness.
Our First Randomized Algorithm: 

Quickselect
The Selection Problem

• Recall from last time: the selection problem is to find the $k$th largest element in an unsorted array.

• Can solve in $O(n \log n)$ time by sorting and taking the $k$th largest element.

• Can solve in $O(n)$ time (with a large constant factor) using the “median-of-medians” algorithm.
Comparison of Selection Algorithms

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Sorting</th>
<th>Median of Medians</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000000</td>
<td>0.92</td>
<td>0.37</td>
</tr>
<tr>
<td>200000000</td>
<td>1.9</td>
<td>0.74</td>
</tr>
<tr>
<td>300000000</td>
<td>2.9</td>
<td>1.05</td>
</tr>
<tr>
<td>400000000</td>
<td>3.94</td>
<td>1.43</td>
</tr>
<tr>
<td>500000000</td>
<td>5.01</td>
<td>1.83</td>
</tr>
<tr>
<td>600000000</td>
<td>6.06</td>
<td>2.12</td>
</tr>
<tr>
<td>700000000</td>
<td>7.16</td>
<td>2.54</td>
</tr>
<tr>
<td>800000000</td>
<td>8.26</td>
<td>2.89</td>
</tr>
<tr>
<td>900000000</td>
<td>9.3</td>
<td>3.2</td>
</tr>
</tbody>
</table>
Partition-Based Selection

- Recall: The median-of-medians algorithm belongs to a family of algorithms based on the partition algorithm:
  - Choose a pivot.
  - Use partition to place it correctly.
  - Stop if the pivot is in the right place.
  - Recurse on one piece of the array otherwise.
- With no constraints on how the pivot is chosen, runtime is $\Omega(n)$ and $O(n^2)$. 
Randomized Selection

• Silly question: What happens if you pick pivots completely at random?
• Intuitively, gives reasonably good probability of picking a good pivot.
• This algorithm is called **quicksselect**.
Analyzing Quickselect

- When analyzing a randomized algorithm, we typically are interested in learning the following:
  - What is the average-case runtime of the function?
  - How likely are we to achieve that average-case runtime?
- We'll answer these questions in a few minutes.
- For now, let's start off with a simpler question...
The Worst Case

- In the worst-case, a partition-based selection algorithm can take \( O(n^2) \) time.
- Recall: What triggers the worst-case behavior of the selection algorithm?
- **Answer:** Continuously pick the largest or smallest element on each iteration.
- Since quickselect picks pivots randomly, what is the probability that this happens in quickselect?
Triggering the Worst Case

- Let $\mathcal{E}_k$ be the event that we pick the largest or smallest element of the array when there are $k$ elements left.

- Let event $\mathcal{E}$ correspond to the worst-case runtime of quickselect occurring.

- We can then define $\mathcal{E}$ as the event

$$\mathcal{E} = \bigcap_{i=1}^{n} \mathcal{E}_i$$

- Question: What is $P(\mathcal{E})$?
Triggering the Worst Case

- We have
  \[ P(\mathcal{E}) = P\left(\bigcap_{i=1}^{n} \mathcal{E}_i\right) \]

- Since all \( \mathcal{E}_i \)'s are independent (we make independent random choices at each level), this simplifies to
  \[ P(\mathcal{E}) = P\left(\bigcap_{i=1}^{n} \mathcal{E}_i\right) = \prod_{i=1}^{n} P(\mathcal{E}_i) \]

- If \( i > 1 \), then \( P(\mathcal{E}_i) = 2 / i \). \( P(\mathcal{E}_1) = 1 \). Thus
  \[ P(\mathcal{E}) = \prod_{i=1}^{n} P(\mathcal{E}_i) = \prod_{i=2}^{n} \frac{2}{i} = \frac{2^{n-1}}{n!} \]
Eensy Weensy Numbers

- The probability of triggering the worst-case behavior of quickselect is

\[ P(\mathcal{E}) = \frac{2^{n-1}}{n!} \]

- To put that in perspective: if \( n = 31 \), then \( 2^{n-1} \approx 10^9 \) and \( n! \approx 8 \times 10^{33} \).

- This is extremely unlikely!
On Average

• We know that the probability of getting a worst-case runtime is vanishingly small.

• But how does the algorithm do on average? Is it $\Theta(n)$? $\Theta(n \log n)$? Something else?

• Totally reasonable thing to do: try running it and see what happens!
## Comparison of Selection Algorithms

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Sorting</th>
<th>Median of Medians</th>
<th>Quickselect</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000</td>
<td>0.92</td>
<td>0.37</td>
<td>0.11</td>
</tr>
<tr>
<td>20000000</td>
<td>1.9</td>
<td>0.74</td>
<td>0.14</td>
</tr>
<tr>
<td>30000000</td>
<td>2.9</td>
<td>1.05</td>
<td>0.27</td>
</tr>
<tr>
<td>40000000</td>
<td>3.94</td>
<td>1.43</td>
<td>0.44</td>
</tr>
<tr>
<td>50000000</td>
<td>5.01</td>
<td>1.83</td>
<td>0.53</td>
</tr>
<tr>
<td>60000000</td>
<td>6.06</td>
<td>2.12</td>
<td>0.64</td>
</tr>
<tr>
<td>70000000</td>
<td>7.16</td>
<td>2.54</td>
<td>0.69</td>
</tr>
<tr>
<td>80000000</td>
<td>8.26</td>
<td>2.89</td>
<td>1.01</td>
</tr>
<tr>
<td>90000000</td>
<td>9.3</td>
<td>3.2</td>
<td>0.72</td>
</tr>
</tbody>
</table>
An Average-Case Analysis

- Our guess: average runtime is $\Theta(n)$.
- How would we go about proving this?
- Since algorithm is recursive, might want to write a recurrence relation.
- This is challenging: the split size isn't guaranteed, so we have no idea how big our subproblems will be!
- Let's try another approach...
An Accounting Trick

- Because quickselect makes at most one recursive call, we can think of the algorithm as a chain of recursive calls:

137 → 96 → 64 → 42 → 13 → 7

- Accounting trick: group multiple calls together into one “phase” of the algorithm.

- The sum of the work done by all calls is equal to the sum of the work done by all phases.

- Goal: Pick phases intelligently to simplify analysis.
Picking Phases

• Let's define one “phase” of the algorithm to be when the algorithm decreases the size of the input array to 75% of the original size or less.

• Why 75%?
  • If array shrinks by any constant factor from phase to phase and only does linear work per phase, total work done is linear.
  • The number 75% has a nice intuition...
Triggering 75% / 25%

• Suppose that we pick a pivot whose value is in the middle 50% of all array values.
• Then 25% of array values are larger and 25% of array values are smaller.
• Guaranteed to get a 75% / 25% split!
• A phase ends as soon as we pick a pivot in the middle 50% of all values.
Analyzing the Runtime

- Number the phases 0, 1, 2, ...
- In phase $k$, the array size is at most $n(3/4)^k$.
- Last phase numbered at most $\lceil \log_{4/3} n \rceil$.
- Let $X_k$ be a random variable equal to the number of recursive calls in phase $k$.
- Work done in phase $k$ is at most $X_k \cdot cn \left(\frac{3}{4}\right)^k$ (for some constant $c$)
- Let $W$ be a random variable denoting the total work done. Then

$$W \leq \sum_{k=0}^{\lceil \log_{4/3} n \rceil} \left( X_k \cdot cn \left(\frac{3}{4}\right)^k \right) = cn \sum_{k=0}^{\lceil \log_{4/3} n \rceil} \left( X_k \left(\frac{3}{4}\right)^k \right)$$
The Average-Case Analysis

• Our goal is to determine the expected runtime for quickselect on an array of size \( n \).
• This is \( E[W] \), the expected value of \( W \).
• This is given by

\[
E[W] \leq E \left[ cn \sum_{k=0}^{\lceil \log_{4/3} n \rceil} \left( X_k \left( \frac{3}{4} \right)^k \right) \right]
\]
Properties of Expectation

• The expected value of a constant or non-random variable is just that constant or variable itself:

\[ E[c] = c \]

• Expected value is a linear operator:

\[ E[aX + b] = aE[X] + b \]

\[ E[X + Y] = E[X] + E[Y] \]

• Note that the second claim holds even if \( X \) and \( Y \) are dependent variables.
Simplifying Our Expression

\[
E[W] \leq E \left[ cn \sum_{k=0}^{\lceil \log_{4/3} n \rceil} X_k \left( \frac{3}{4} \right)^k \right] \\
= cn \cdot E \left[ \sum_{k=0}^{\lceil \log_{4/3} n \rceil} X_k \left( \frac{3}{4} \right)^k \right] \\
= cn \cdot \sum_{k=0}^{\lceil \log_{4/3} n \rceil} E \left[ X_k \left( \frac{3}{4} \right)^k \right]
\]
\[ E[X_k] \]

- By definition:
  \[
  E[X_k] = \sum_{i=0}^{\infty} i \cdot P(X_k = i)
  \]

Recall: \( X_k \) is the number of calls within phase \( k \).

- Equivalently: The number of calls before a pivot is chosen in the middle 50% of the elements.

- Can we determine this explicitly?
\[ E[X_k] \]

- \( E[X_k] \) is defined by
  \[ E[X_k] = \sum_{i=0}^{\infty} i \cdot P(X_k = i) \]
- \( P(X_k = i) \) is the probability that the first \( i - 1 \) pivots we chose weren't in the middle 50% and that the \( i \)th pivot is in the middle 50%.
  - (As an edge case, it's 0 when \( i = 0 \).)
- As a simplification: assume that whenever we pick a pivot, we can choose from any of the \( n \) elements present at the start of the phase.
- Only makes it harder to end the phase; provides an upper bound on the phase length.
\textbf{E}[X_k]}

- Recall: \( E[X_k] \) is defined by
  \[
  E[X_k] = \sum_{i=0}^{\infty} i \cdot P(X_k = i)
  \]

- Under the assumption that all pivot choices are independent, \( P(X_k = i) \) is given by
  \[
  P(X_k = i) = \left(\frac{1}{2}\right)^i
  \]

- Probability the first \( i - 1 \) pivots are in the outer 50% and the \( i \)th pivot was in the inner 50%.

- Therefore
  \[
  E[X_k] = \sum_{i=0}^{\infty} i \cdot P(X_k = i) \leq \sum_{i=1}^{\infty} \frac{i}{2^i} = 2
  \]
Finalizing the Computation

\[ E[W] \leq c n \cdot \sum_{k=0}^{\lfloor \log_{4/3} n \rfloor} E[X_k] \left( \frac{3}{4} \right)^k \]

\[ \leq c n \cdot \sum_{k=0}^{\lfloor \log_{4/3} n \rfloor} 2 \left( \frac{3}{4} \right)^k \]

\[ = 2 c n \cdot \sum_{k=0}^{\lfloor \log_{4/3} n \rfloor} \left( \frac{3}{4} \right)^k \]

\[ \leq 2 c n \cdot \sum_{k=0}^{\infty} \left( \frac{3}{4} \right)^k \]

\[ = 8 c n \]

\[ = O(n) \]
Bounding the Spread

- We now know that quickselect runs in expected $O(n)$ time.
- How *likely* is it that the runtime is $O(n)$?
Bounding the Spread

- Idea: Devise a formula for the probability that every phase terminates within $r$ steps.
- If this happens, quickselect will run in time

$$c n \cdot \sum_{k=0}^{\lceil \log_{4/3} n \rceil} X_k \left( \frac{3}{4} \right)^k \leq c n \cdot \sum_{k=0}^{\lceil \log_{4/3} n \rceil} r \left( \frac{3}{4} \right)^k$$

$$= c n r \cdot \sum_{k=0}^{\lceil \log_{4/3} n \rceil} \left( \frac{3}{4} \right)^k$$

$$\leq 4 c n r$$

$$= O(n r)$$

- Goal: Find the probability (as a function of $r$) that this occurs.
Bounding the Spread

• We want the probability of the event

   **All phases terminate within \( r \) steps.**

• Mathematically, it's easier to work with the probability of the *complement* of this event:

   **At least one phase terminates in at least \( r + 1 \) steps.**

• We can compute the probability of the first event by subtracting the probability of the second event from one.
Long Phase Runtimes

• The probability that phase $k$ takes more than $r$ steps to finish is given by

\[
P(X_k > r)
\]

• This is

\[
P(X_k > r) = P\left( \bigcup_{i=r+1}^{\infty} X_k = i \right)
\]

• Since these events are all mutually exclusive:

\[
P(X_k > r) = \sum_{i=r+1}^{\infty} P(X_k = i) \leq \sum_{i=r+1}^{\infty} \frac{1}{2^i} = \frac{1}{2^r}
\]
Long Phase Runtimes

- The probability that any phase takes more than $r$ steps to finish is

$$P\left( \bigcup_{i=0}^{\lceil \log_{4/3} n \rceil} X_i > r \right)$$

- These are not mutually exclusive events – we may have multiple different phases finish in more than $r$ steps.

- We can use the union bound to get an upper-bound on the true value:

$$P\left( \bigcup_{i=0}^{\infty} \mathcal{E}_i \right) \leq \sum_{i=0}^{\infty} P(\mathcal{E}_i)$$
Long Phase Runtimes

• The probability that *any* phase takes more than \( r \) steps to finish is

\[
P\left( \bigcup_{i=0}^{\lfloor \log_{4/3} n \rfloor} X_i > r \right)
\]

• Using the union bound:

\[
P\left( \bigcup_{i=0}^{\lfloor \log_{4/3} n \rfloor} X_i > r \right) \leq \sum_{i=0}^{\lfloor \log_{4/3} n \rfloor} P(X_i > r) \leq \sum_{i=0}^{\lfloor \log_{4/3} n \rfloor} \frac{1}{2^r} = \frac{\lfloor \log_{4/3} n \rfloor + 1}{2^r}
\]
Long Phase Runtimes

• For any number $r$, the probability that any one phase takes more than $r$ steps to finish is

$$\frac{\lceil \log_{4/3} n \rceil + 1}{2^r}$$

• If for any value $s$ we pick $r = s + \log_2(\lceil \log_{4/3} n \rceil + 1)$, then the probability that any phase takes more than $r$ steps to complete is at most

$$\frac{\lceil \log_{4/3} n \rceil + 1}{2^{s \log_2(\lceil \log_{4/3} n \rceil + 1)}} = \frac{\lceil \log_{4/3} n \rceil + 1}{2^s(\lceil \log_{4/3} n \rceil + 1)} = \frac{1}{2^s}$$
Bounding the Runtime

- Recall: If all phases terminate within $r$ steps, the total runtime will be $O(nr)$.
- If we pick $r = s + \log_2([\log_{4/3} n] + 1)$, then the runtime will be $O(ns + n \log \log n)$ with probability at least $1 - 1/2^s$.
- For any constant $k$, pick $s = \log_2 n^k = k \log_2 n$. Probability that the runtime is $O(n \log n)$ is at least $1 - 1 / n^k$.
- **Definition:** Event $\mathcal{E}$ occurs with high probability iff $P(\mathcal{E}) \geq 1 - 1 / n^c$ for some $c \geq 1$.
- Quickselect runs in time at most $O(n \log n)$ with high probability.
Wrap-Up: Introselect
Where We Stand

- The median-of-medians algorithm has runtime $O(n)$, but has a large constant factor.
- Quickselect has average-case runtime $O(n)$ with a low constant factor, but isn't guaranteed to run in time $O(n)$.
- Can we get the best of both worlds?
Introspective Selection

- The **introselect** algorithm intelligently combines median-of-medians and quickselect.
- Idea: Run quickselect, but keep track of how many iterations have passed in the current phase.
- If the phase ends before the number of iterations exceeds some constant $k$, reset the counter and continue.
- Otherwise, run the median-of-medians algorithm to choose a pivot and reset the counter.
Introspective Selection

• Assuming introselect makes good random choices, it is inappreciably slower than normal quickselect.

• If it makes too many bad choices, we do some expensive median-of-medians steps, which is slower but ensures linear time.

• Net result is an algorithm that has worst-case $O(n)$ runtime and on expectation matches quickselect's runtime.
## Comparison of Selection Algorithms

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Sorting</th>
<th>Median of Medians</th>
<th>Quickselect</th>
<th>Introselect</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000000</td>
<td>0.92</td>
<td>0.37</td>
<td>0.11</td>
<td>0.07</td>
</tr>
<tr>
<td>200000000</td>
<td>1.9</td>
<td>0.74</td>
<td>0.14</td>
<td>0.17</td>
</tr>
<tr>
<td>300000000</td>
<td>2.9</td>
<td>1.05</td>
<td>0.27</td>
<td>0.17</td>
</tr>
<tr>
<td>400000000</td>
<td>3.94</td>
<td>1.43</td>
<td>0.44</td>
<td>0.33</td>
</tr>
<tr>
<td>500000000</td>
<td>5.01</td>
<td>1.83</td>
<td>0.53</td>
<td>0.42</td>
</tr>
<tr>
<td>600000000</td>
<td>6.06</td>
<td>2.12</td>
<td>0.64</td>
<td>0.41</td>
</tr>
<tr>
<td>700000000</td>
<td>7.16</td>
<td>2.54</td>
<td>0.69</td>
<td>0.51</td>
</tr>
<tr>
<td>800000000</td>
<td>8.26</td>
<td>2.89</td>
<td>1.01</td>
<td>0.56</td>
</tr>
<tr>
<td>900000000</td>
<td>9.3</td>
<td>3.2</td>
<td>0.72</td>
<td>0.88</td>
</tr>
</tbody>
</table>
Next Time

- Quicksort
- Indicator Random Variables
- Harmonic Numbers