Outline:

- Quicksort
  - Average-Case Analysis of QuickSort
  - Randomized quicksort
- Selection
  - The selection problem
  - First solution: Selection by sorting
  - Randomized Selection
Quicksort($A, p, r$)

begin
    if $p < r$ then
        $q =$ Partition($A, p, r$);
        Quicksort($A, p, q - 1$);
        Quicksort($A, q + 1, r$);
    end
end

- **Partition($A, p, r$)** reorders items in $A[p \ldots r]$;
  items $< A[r]$ are to its left; items $> A[r]$ to its right.

- Showed that if input is a random input (permutation) of $n$ items, then average running time is $O(n \log n)$.
Formally, the average running time can be defined as follows:

- \( I_n \) is the set of all \( n! \) inputs of size \( n \)
- \( I \in I_n \) is any particular size-\( n \) input
- \( R(I) \) is the running time of the algorithm on input \( I \)

Then, the average running time over the random inputs is

\[
\sum_{I \in I_n} \Pr(I)R(I) = \frac{1}{n!} \sum_{I \in I_n} R(I) = O(n \log n)
\]

Only fact that was used was that \( A[r] \) was a random item in \( A[p \ldots r] \), i.e., the partition item is equally likely to be any item in the subset.
Outline:

- Quicksort
  - Average-Case Analysis of QuickSort
  - Randomized Quicksort

- Selection
  - The selection problem
  - First solution: Selection by sorting
  - Randomized Selection
Randomized-Partition\( (A, p, r) \)

Idea:

- In the algorithm Partition\( (A, p, r) \), \( A[r] \) is always used as the pivot \( x \) to partition the array \( A[p..r] \).
- In the algorithm Randomized-Partition\( (A, p, r) \), we randomly choose \( j, p \leq j \leq r \), and use \( A[j] \) as pivot.
- Idea is that if we choose randomly, then the chance that we get unlucky every time is extremely low.
Let \( \text{random}(p, r) \) be a pseudorandom-number generator that returns a random number between \( p \) and \( r \)

**Randomized-Partition**\((A, p, r)\)

\[
\begin{align*}
\text{begin} & \\
\quad j &= \text{random}(p, r); \\
\quad \text{exchange } A[r] \text{ and } A[j]; \\
\quad \text{Partition}(A, p, r);
\end{align*}
\]

\text{end}
We make use of the Randomized-Partition idea to develop a new version of quicksort

Randomized-Quicksort($A, p, r$)

begin
  if $p < r$ then
    $q = \text{Randomized-Partition}(A, p, r)$;
    Randomized-Quicksort($A, p, q - 1$);
    Randomized-Quicksort($A, q + 1, r$);
  end
end
Let $I \in I_n$ be any input.

- The running time $R(I)$ depends upon the random choices made by the algorithm in the step
  $$\text{random}(p, r); \ \text{exchange} \ A[r] \ \text{and} \ A[j]$$

- This can be different for different random choices.

- We are actually interested in $E(R(I))$, the \textit{Expected (average) Running Time (ERT)}
  
  \begin{itemize}
  \item average now is \textbf{not over the input}, which is fixed
  \item average is \textbf{over the random choices made by the algorithm}.
  \end{itemize}
Let $I \in \mathcal{I}_n$ be any input.

Want $E(R(I))$, the Expected Running Time, where average is taken over random choices of algorithm.

Suprisingly, we can use almost exactly the same analysis that we used for the average-case analysis of Quicksort. Recall that only facts that we used were

- Item used as a pivot is random among all items
- this statement is true in all subproblems as well.

Those two facts are still valid here, so the expected running time still satisfies

$$C_n = n - 1 + \frac{1}{n} \sum_{1 \leq k \leq n} (C_{k-1} + C_{n-k})$$

which we already proved was $O(n \log n)$. 
Running Time of Randomized-Quicksort

- Just saw that for any fixed input of size $n$, ERT is $O(n \log n)$

- Randomized Quicksort is a Randomized Algorithm
  - Makes Random choices to determine what algorithm does next
  - When rerun on same input, algorithm can make different choices and have different running times
  - Running time of Randomized Algorithm is worst case ERT over all inputs $I$. In our case

  $$\max_{I \in I_n} E[R(I)] = O(n \log n)$$

- Contrast with Average Case Analysis
  - When rerun on same input, algorithm always does same things, so $R(i)$ is deterministic.
  - Given a probability distribution on inputs, calculate average running time of algorithm over all inputs

  $$\sum_{I \in I_n} \Pr(I) R(I)$$
Outline:

- Quicksort
  - Average-Case Analysis of QuickSort
  - Randomized Quicksort
- Selection
  - The Selection problem
  - First solution: Selection by sorting
  - Randomized Selection
The Selection Problem

**Definition (Selection Problem)**

Given a sequence of numbers \(\langle a_1, \ldots, a_n \rangle\), and an integer \(i\), \(1 \leq i \leq n\), find the \(i\)th smallest element. When \(i = \lceil n/2 \rceil\), this is called the median problem.

**Example**

Given \(\langle 1, 8, 23, 10, 19, 33, 100 \rangle\), the 4th smallest element is 19.

**Question**

How can this problem be solved efficiently?
Outline:

- Quicksort
  - Average-Case Analysis of QuickSort
  - Randomized quicksort
- Selection
  - The Selection problem
  - First solution: Selection by sorting
  - Randomized Selection
First Solution: Selection by Sorting

1. Sort the elements in ascending order with any algorithm of complexity $O(n \log n)$.
2. Return the $i$th element of the sorted array.

The complexity of this solution is $O(n \log n)$

Question
Can we do better?

Answer: YES, by using Randomized-Partition($A, p, r$)!
Outline:
- Quicksort
  - Average-Case Analysis of QuickSort
  - Randomized quicksort
- Selection
  - The Selection problem
  - First solution: Selection by sorting
  - Randomized Selection
**Problem:** Select the $i$th smallest element in $A[p..r]$, where $1 \leq i \leq r - p + 1$

**Solution:** Apply Randomized-Partition($A, p, r$), getting

$p \quad q \quad r$

$k = q - p + 1$

1. $i = k$
   - pivot is the solution
2. $i < k$
   - the $i$th smallest element in $A[p..r]$ must be the $i$th smallest element in $A[p..q - 1]$
3. $i > k$
   - the $i$th smallest element in $A[p..r]$ must be the $(i - k)$th smallest element in $A[q + 1..r]$

If necessary, **recursively** call the same procedure to the subarray.
Randomized-Select($A, p, r, i$), $1 \leq i \leq r - p + 1$

```plaintext
if $p = r$ then
    | return $A[p]$
end

$q = \text{Randomized-Partition}(A, p, r)$;
$k = q - p + 1$;
if $i = k$ then return $A[q]$;
// the pivot is the answer
else if $i < k$ then
    | return Randomized-Select($A, p, q - 1, i$)
else
    | return Randomized-Select($A, q + 1, r, i - k$)
end
```

To find the $i$th smallest element in $A[1..n]$, call
Randomized-Select($A, 1, n, i$)
Recall that if pivot $q$ is $k$th item in order, then algorithm is

If $i = k$, stop.  
If $i < k \Rightarrow A[p..q-1]$.  
If $i > k \Rightarrow A[q+1..r]$.

Let $m = p - r + 1$.

Note that if $k = p + \lfloor \frac{m}{2} \rfloor$ was always true, this would halve the problem size at every step and the running time would be at most

$$n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \ldots = n \left( 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} \right) \leq 2n$$

This isn’t a realistic analysis because $q$ is chosen randomly, so $k$ is actually random number between $p..r$. 

Running Time of Randomized-Select($A, 1, n, i$)
Recall that if pivot $q$ is $k$th item in order then algorithm is

If $i = k$, stop.  
If $i < k \Rightarrow A[p..q−1]$.  
If $i > k \Rightarrow A[q+1..r]$.

Let $m = p − r + 1$.

Suppose that we could guarantee that $p + \frac{m}{4} \leq k \leq p + \frac{3}{4}m$.

This would be enough to force linearity because the recursive call would always be to a subproblem of size $\leq \frac{3}{4}m$ and the running time of the entire algorithm would be at most

$$n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)^3 n + \ldots \leq 4n$$
Set $m = p - r + 1$. We saw that if

$$p + \frac{m}{4} \leq k \leq p + \frac{3}{4}m$$

then algorithm is linear.

While this is not always true, we can easily see that

$$\Pr \left( p + \frac{m}{4} \leq k \leq p + \frac{3}{4}m \right) \geq \frac{1}{2}.$$

This means that each stage of the algorithm has probability at least $1/2$ of reducing the problem size by $3/4$. A careful analysis will show that this implies an $O(n)$ expected running time.
Running Time of Randomized-Select($A, 1, n, i$)

More formally, suppose $t$’th call to the algorithm is $A(p_t, r_t, i_t)$. Let $M_t = r_t - p_t + 1$ be size of array in the subproblem and $k_t$ location of the random pivot in that subarray. Note

- $p_1 = 1$, $r_1 = n$, $M_1 = n$
- $M_{t+1} \leq M_t - 1$
- Total cost of the algorithm is bounded by $\sum_t M_t$
- Set $E_t$ to be event that is true if

$$p_t + \frac{M_t}{4} \leq k_t \leq p_t + \frac{3}{4}M_t,$$

and false otherwise. Then

- $\Pr(E_t) \geq 1/2$
- If $E_t$ occurs then $M_{t+1} \leq \frac{3}{4}M_t$. 
Recall that
\[ M_1 = n; \quad M_{t+1} \leq M_t - 1; \quad \text{If } E_t \Rightarrow M_{t+1} \leq \frac{3}{4} M_t. \]

Note that \( E_t \) is undefined after the algorithm ends, i.e., \( M_t \leq 1 \). For larger \( t \), define \( E_t \) by flipping fair coin and setting \( E_t \) True if HEAD seen.

Now define \( M'_t \) as follows

- \( M'_1 = n \)
- \( \text{If } E_t \Rightarrow M'_{t+1} = \frac{3}{4} M'_t. \quad \text{If } (\text{not } E_t) \Rightarrow M'_{t+1} = M'_t. \)

Then \( \forall t, \quad M_t \leq M'_t. \)

In particular, since \( \sum_t M_t \) bounds the algorithm’s runtime, \( \sum_t M'_t \) also bounds the algorithm’s runtime!
Consider a $p$-biased coin, i.e., a coin with probability $p$ of turning up Heads and $(1 - p)$ of Tails.

- Let $X$ be the number of flips until seeing the first Head.
- $X$ is a *Geometric Random Variable* with parameter $p$.
- $\Pr(X = i) = (1 - p)^{i-1} p$
- $E(X) = \frac{1}{p}$
- In particular, if the coin is fair, i.e., $p = 1/2$, then $E(X) = 2$.
- If at every step the coin probability can change, BUT the probability of Heads is always $\geq 1/2$, then $E(X) \leq 2$.
- In this case we say $X$ is *bounded* by a geometric random variable with $p = 1/2$. 
Running Time of Randomized-Select($A, 1, n, i$)

Given sequence of events $E_1, E_2, E_3, \ldots$ with $\forall t, \Pr(E_t) \geq 1/2$

- Set $Z_0 = 1$ and $Z_i$ to be the location of the $i^{th}$ true $E_t$.
- Set $X_i = Z_{i+1} - Z_i$.
  - $X_i$ is time from $Z_i$ until next success so it is bounded by a geometric random variable with $p = 1/2$.
  - $\Rightarrow$ Then $E(X_i) \leq 2$

- Recall $M_1 = n$; If $E_t$, set $M_{t+1} = \frac{3}{4} M_t$. Else $M_{t+1} = M_t$.

  Then $\sum_t M'_t = \sum_i X_i \left(\frac{3}{4}\right)^i n$ (why)

- By linearity of expectation

$$E \left( \sum_t M'_t \right) = \sum_i E(X_i) \left(\frac{3}{4}\right)^i n \leq 2n \sum_i \left(\frac{3}{4}\right)^i = 8n$$

QED
Worst Case:

\[ T(n) = n - 1 + T(n - 1), \quad T(n) = O(n^2). \]

Expected Running Time:

\[ O(n) \]

Expected running time much better than worst case!
Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Answer:

- Randomized Selection needs to work on only one of the two subproblems.
- Randomized Quicksort needs to work on both of the two subproblems.
How do we generate a random number?
Dice, coin flipping, roulette wheels, ...

How does a computer generate a random number?
- By hardware: electronic noise, thermal noise, etc. Expensive but “true” random numbers in some sense
- By software: pseudorandom numbers. A long sequence of seemingly random numbers whose pattern is difficult to find
- Pseudorandom numbers are good enough for most applications
Another Analysis of the Running Time of Randomized-Select($A, 1, n, i$)

$T(n)$: upper bound on the expected number of comparisons made by Randomized-Select($A, 1, n, i$) for any $i$

$T(1) = 0$

For $n > 1$, we get

$T(n) \leq n + \sum_{k=1}^{n} \left( \frac{1}{n} \cdot T(\max\{k - 1, n - k\}) \right)$

initial partition recursion, assume the bad case

Which is a complicated recurrence!

We use the guess & induction method

Guess:

$T(n) \leq c \cdot n$, for all $n$

for some constant $c$ to be figured out later.
Proof that $T(n) \leq cn$

Induction step: Assume that $T(m) \leq cm$ for all $m \leq n - 1$. Then try to show $T(n) \leq cn$:

$$T(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k)$$

$$\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck$$

$$\leq \ldots$$

$$\leq \frac{3c}{4} n + \frac{c}{2} + n$$

We want $\frac{3c}{4} n + \frac{c}{2} + n \leq cn$, or $n \geq \frac{2c}{c-4}$.

If we choose $c \geq 12$. Then the induction step works for $n \geq 3$.

Induction basis: $T(1) \leq c \cdot 1$, $T(2) \leq c \cdot 2$.

So if we choose $c = \max\{12, T(1), T(2)/2\}$, then the entire proof works.