CSE 202: Design and Analysis of Algorithms

Lecture 15

Instructor: Kamalika Chaudhuri
Announcements

• Final will be take-home. Given out at the end of class on June 1st, due June 3rd, 4pm

• No collaboration on the final!

• TA office hours on Thu May 19 from 12:30-1:30pm in room B275

• HW3 due in class on Wed
Last Class: Randomized Algorithms

- Algorithm can make random decisions
- Why randomized algorithms? Simple and efficient
- Examples: Symmetry-breaking, graph algorithms, quicksort, hashing, load balancing, cryptography, etc
Last Class: Randomized Algorithms

- Contention Resolution
- Some Facts about Random Variables
Last Class: Expectation, Linearity of Expectation

Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot \Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

**Linearity of Expectation:** $E[X + Y] = E[X] + E[Y]

**Examples:** Guessing a card, birthday paradox, coupon collector
Last Class: Variance and Properties

Given a random variable $X$, its variance $\text{Var}[X]$ is defined as:

$$\text{Var}(X) = E[(X - E[X])^2]$$

Properties of Variance:

1. $\text{Var}(X) = E[X^2] - (E[X])^2$
2. If $X$ and $Y$ are independent random variables, then, $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$
3. For any constants $a$ and $b$, $\text{Var}(aX + b) = a^2 \text{Var}(X)$

Variance of a random variable measures its “spread”
Randomized Algorithms

- Contention Resolution
- Some Facts about Random Variables
- Global Minimum Cut Algorithm
Global Min-Cut

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Example:**

![Graph G](image1)

A global min-cut is not the same as an s-t min cut.

How can we find a global min-cut using $n - 1$ max-flows?

We can do better for the global min-cut.
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

- #edges to pick from = 14
- Pick $(b, f)$ (probability $1/14$)
Problem: Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.

Example:

- #edges to pick from = 13
- Pick $(g, h)$ (probability $1/13$)
**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u$, $v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

![Graph diagram]

#edges to pick from = 12
Pick $(d, gh)$ (probability $1/6$)
Problem: Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.

Example:

#edges to pick from = 10
Pick $(a, e)$ (probability $1/10$)
**Karger’s Min-Cut Algorithm**

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

```
#edges to pick from = 9
Pick (ae, bf) (probability 4/9)
```
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

- #edges to pick from = 5
- Pick $(c, dgh)$ (probability 3/5)
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.

**Example:**

Done! Output $(aebf, cdgh)$.

**Original Graph:**
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Proof: Total degree of $n$ nodes $= 2|E|$, $n$ nodes in total
Karger's Algorithm: Analysis

**Karger’s Min-Cut Algorithm:**
1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Fact 1.** If there are $n$ nodes, then the average degree of a node is $2|E|/n$

**Fact 2.** The minimum cut size is at most $2|E|/n$

**Proof:** From Fact 1, there is at least one node $x$ with degree at most $2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge e = (u, v) in E uniformly at random. Collapse u and v into a single
   node (allowing multiple edges)
2. Let u, v be the nodes. Output (U, V - U), where U = {nodes that went into u}

Fact 1. If there are n nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$
Proof: From Fact 1, there is at least one node x with degree at most $2|E|/n$
The cut $(x, V - x)$ has $\text{deg}(x)$ edges. The minimum cut size is thus at most $\text{deg}(x) \leq 2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$

Proof: Follows directly from Fact 2
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge \( e = (u, v) \) in \( E \) uniformly at random. Collapse \( u \) and \( v \) into a single node (allowing multiple edges)
2. Let \( u, v \) be the nodes. Output \( (U, V - U) \), where \( U = \{ \text{nodes that went into } u \} \)

Fact 1. If there are \( n \) nodes, then the average degree of a node is \( 2|E|/n \)
Fact 2. The minimum cut size is at most \( 2|E|/n \)
Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most \( 2/n \)
Observe: Bad case is when the algorithm selects an edge \( e \) across the min-cut
**Karger’s Algorithm: Analysis**

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Fact 1.** If there are $n$ nodes, then the average degree of a node is $2|E|/n$

**Fact 2.** The minimum cut size is at most $2|E|/n$

**Fact 3.** If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$

Observe: Bad case is when the algorithm selects an edge $e$ across the min-cut

$\Pr[\text{Output cut } = \text{min-cut}] = \Pr[\text{First selected edge not in min-cut}] \times \Pr[\text{Second selected edge not in min-cut}] \times \ldots$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single
   node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies
   across the min cut is at most $2/n$

Observe: Bad case is when the algorithm selects an edge $e$ across the min-cut

Pr[ Output cut = min-cut ] = Pr[First selected edge not in min-cut] x Pr[Second
   selected edge not in min-cut] x ....

$$\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{3}\right)$$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge \( e = (u, v) \) in \( E \) uniformly at random. Collapse \( u \) and \( v \) into a single node (allowing multiple edges)
2. Let \( u, v \) be the nodes. Output \((U, V - U)\), where \( U = \{ \text{nodes that went into } u \} \)

**Fact 1.** If there are \( n \) nodes, then the average degree of a node is \( 2|E|/n \)

**Fact 2.** The minimum cut size is at most \( 2|E|/n \)

**Fact 3.** If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most \( 2/n \)

Observe: Bad case is when the algorithm selects an edge \( e \) across the min-cut

\[
\Pr[ \text{Output cut } = \text{min-cut }] = \Pr[\text{First selected edge not in min-cut}] \times \Pr[\text{Second selected edge not in min-cut}] \times \ldots
\]

\[
\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{3}\right)
\]

\[
= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \ldots \frac{2}{4} \cdot \frac{1}{3}
\]
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge e = (u, v) in E uniformly at random. Collapse u and v into a single node (allowing multiple edges)
2. Let u, v be the nodes. Output (U, V - U), where U = \{nodes that went into u\}

Fact 1. If there are n nodes, then the average degree of a node is 2|E|/n
Fact 2. The minimum cut size is at most 2|E|/n
Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most 2/n

Observe: Bad case is when the algorithm selects an edge e across the min-cut

Pr[ Output cut = min-cut ] = Pr[First selected edge not in min-cut] x Pr[Second selected edge not in min-cut] x ....

\[
\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{3}\right)
\]

\[
= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)}
\]
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$
Fact 2. The minimum cut size is at most $2|E|/n$
Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$

Observe: Bad case is when the algorithm selects an edge $e$ across the min-cut

$$
\Pr[\text{Output cut} = \text{min-cut}] = \Pr[\text{First selected edge not in min-cut}] \times \Pr[\text{Second selected edge not in min-cut}] \times \ldots \\
\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{3}\right) \\
= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)}
$$

Thus, outputs min-cut w.p. $2/n^2$; can run it $O(n^2)$ times and pick the best of the outputs
Randomized Algorithms

- Contention Resolution
- Some Facts about Random Variables
- Global Minimum Cut Algorithm
Randomized Algorithms: Two Types

**Monte Carlo Algorithm:**
Always has the same running time
Not **guaranteed** to return the correct answer (returns a correct answer only with some probability)

**Las Vegas Algorithm:**
Always guaranteed to return the correct answer
Running time fluctuates (probabilistically)
Randomized Algorithms: Two Types

**Monte Carlo Algorithm:**
Always has the same running time
Not **guaranteed** to return the correct answer (returns a correct answer only with some probability)

**Las Vegas Algorithm:**
Always guaranteed to return the correct answer
Running time fluctuates (probabilistically)

**Fact:** Suppose a Monte Carlo algorithm succeeds w.p. \( p \). Then, it can be made to succeed w.p. \( 1 - t \) for any (small) \( t \) by running it \( O(\log (1/t)/p) \) time
Randomized Algorithms: Two Types

Monte Carlo Algorithm:
Always has the same running time
Not guaranteed to return the correct answer (returns a correct answer only with some probability)

Las Vegas Algorithm:
Always guaranteed to return the correct answer
Running time fluctuates (probabilistically)

Fact: Suppose a Monte Carlo algorithm succeeds w.p. p. Then, it can be made to succeed w.p. 1 - t for any (small) t by running it $O(\log (1/t)/p)$ time

Proof: Suppose we run the algorithm k times. Then,

Pr[Algorithm is wrong every time] = $(1 - p)^k < t$
when $k = O(\log (1/t)/p)$
Randomized Algorithms

• Contention Resolution
• Some Facts about Random Variables
• Global Minimum Cut Algorithm
• Randomized Selection and Sorting
Computing Percentiles

An f-th percentile is the value below which f percent of observations fall.

Given array $A[1..n]$, find the $k$-th smallest element in $A$.

**Example:** Median = $n/2$-th smallest element = 50th percentile
How to compute the median in $O(n \log n)$ time?
Randomized Selection

Given array A[1..n], find the k-th smallest element in A

A Divide and Conquer Algorithm:
Select(A, k)
1. Pick an item v in A
2. Let:
   \( A_L = \) all elements in A that are < v
   \( A_M = \) all elements in A that are = v
   \( A_R = \) all elements in A that are > v
3. Return:
Randomized Selection

Given array A[1..n], find the **k-th smallest element** in A

A Divide and Conquer Algorithm:
Select(A, k)
1. Pick an item v in A
2. Let:
   \( A_L = \) all elements in A that are < v
   \( A_M = \) all elements in A that are = v
   \( A_R = \) all elements in A that are > v
3. Return:

Example:
\[
A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix} \quad v = 5
\]
\[
A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \quad A_M = \begin{bmatrix} 5 & 5 \end{bmatrix} \quad A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix}
\]
Randomized Selection

Given array $A[1..n]$, find the **k-th smallest element** in $A$

**A Divide and Conquer Algorithm:**

Select($A, k$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L$ = all elements in $A$ that are $<$ $v$
   - $A_M$ = all elements in $A$ that are $=$ $v$
   - $A_R$ = all elements in $A$ that are $>$ $v$
3. Return:
   - Select($A_L, k$) if $k \leq |A_L|$

**Example:**

$A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix}$

$v = 5$

$A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix}$

$A_M = \begin{bmatrix} 5 & 5 \end{bmatrix}$

$A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix}$
Given array $A[1..n]$, find the **k-th smallest element** in $A$

**A Divide and Conquer Algorithm:**

Select($A$, $k$)

1. Pick an item $v$ in $A$
2. Let:
   
   $A_L = \text{all elements in } A \text{ that are } < v$
   $A_M = \text{all elements in } A \text{ that are } = v$
   $A_R = \text{all elements in } A \text{ that are } > v$
3. Return:

   Select($A_L$, $k$) if $k \leq |A_L|$
   $v$ if $|A_L| < k \leq |A_L| + |A_M|$

**Example:**

$A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix}$

$v = 5$

$A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix}$

$A_M = \begin{bmatrix} 5 & 5 \end{bmatrix}$

$A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix}$
Randomized Selection

Given array A[1..n], find the **k-th smallest element** in A

A Divide and Conquer Algorithm:

Select(A, k)
1. Pick an item v in A
2. Let:
   - $A_L$ = all elements in A that are < v
   - $A_M$ = all elements in A that are = v
   - $A_R$ = all elements in A that are > v
3. Return:
   - Select($A_L$, k) if $k \leq |A_L|$  
   - v if $|A_L| < k \leq |A_L| + |A_M|$  
   - Select($A_R$, k - |$A_L$| - |$A_M$|) otherwise

Example:

$$A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix} \quad v = 5$$

$$A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \quad A_M = \begin{bmatrix} 5 & 5 \end{bmatrix} \quad A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix}$$
Randomized Selection

Given array A[1..n], find the k-th smallest element in A

A Divide and Conquer Algorithm:

Select(A, k)
1. Pick an item v in A
2. Let:
   \( A_L \) = all elements in A that are < v
   \( A_M \) = all elements in A that are = v
   \( A_R \) = all elements in A that are > v
3. Return:
   Select(\( A_L \), k)                     if k <= |\( A_L \) |
   v                                     if |\( A_L \) | < k <= |\( A_L \) | + |\( A_M \) |
   Select(\( A_R \), k - |\( A_L \) | - |\( A_M \) |)  otherwise

Example:

\[ A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix}, \quad v = 5 \]

\[ A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix}, \quad A_M = \begin{bmatrix} 5 & 5 \end{bmatrix}, \quad A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix} \]
Randomized Selection

Given array A[1..n], find the **k-th smallest element** in A

**A Divide and Conquer Algorithm:**

Select(A, k)
1. Pick an item v in A
2. Let:
   - \( A_L \) = all elements in A that are < v
   - \( A_M \) = all elements in A that are = v
   - \( A_R \) = all elements in A that are > v
3. Return:
   - Select(\( A_L \), k) if \( k \leq |A_L| \)
   - v if \( |A_L| < k \leq |A_L| + |A_M| \)
   - Select(\( A_R \), k - |A_L| - |A_M|) otherwise

**Worst case:** v = smallest or largest element
Randomized Selection

Given array A[1..n], find the \textbf{k-th smallest element} in A

\begin{itemize}
  \item A Divide and Conquer Algorithm:
  \begin{enumerate}
  \item Pick an item \(v\) in \(A\)
  \item Let:
  \begin{align*}
  A_L &= \text{all elements in } A \text{ that are } < v \\
  A_M &= \text{all elements in } A \text{ that are } = v \\
  A_R &= \text{all elements in } A \text{ that are } > v
  \end{align*}
  \item Return:
  \begin{align*}
  \text{Select}(A_L, k) & \quad \text{if } k \leq |A_L| \\
  v & \quad \text{if } |A_L| < k \leq |A_L| + |A_M| \\
  \text{Select}(A_R, k - |A_L| - |A_M|) & \quad \text{otherwise}
  \end{align*}
  \end{enumerate}
\end{itemize}

\textbf{Worst case:} \(v = \) smallest or largest element

Time \(T(n) = O(n) \text{ (for splitting)} + T(n-1)\)
Randomized Selection

Given array \(A[1..n]\), find the **k-th smallest element** in \(A\)

**A Divide and Conquer Algorithm:**

Select\((A, k)\)
1. Pick an item \(v\) in \(A\)
2. Let:
   \(A_L = \) all elements in \(A\) that are \(< v\)
   \(A_M = \) all elements in \(A\) that are \(= v\)
   \(A_R = \) all elements in \(A\) that are \(> v\)
3. Return:
   \[\begin{align*}
   &\text{Select}(A_L, k) \quad \text{if } k \leq |A_L| \\
   &v \quad \text{if } |A_L| < k \leq |A_L| + |A_M| \\
   &\text{Select}(A_R, k - |A_L| - |A_M|) \quad \text{otherwise}
   \end{align*}\]

How to select \(v\)?
Pick \(v\) uniformly at random in \(1..n\)

**Worst case:** \(v = \) smallest or largest element
Time \(T(n) = O(n)\) (for splitting) + \(T(n-1)\)
Solving the recurrence, \(T(n) = O(n^2)\)
Randomized Selection

Given array $A[1..n]$, find the **k-th smallest element** in $A$

**A Divide and Conquer Algorithm:**

Select($A, k$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L = \text{all elements in } A \text{ that are } < v$
   - $A_M = \text{all elements in } A \text{ that are } = v$
   - $A_R = \text{all elements in } A \text{ that are } > v$
3. Return:
   - Select($A_L, k$) if $k \leq |A_L|$
   - $v$ if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R, k - |A_L| - |A_M|$) otherwise

**Worst case:** $v = \text{smallest or largest element}$

Time $T(n) = O(n)$ (for splitting) + $T(n-1)$

Solving the recurrence, $T(n) = O(n^2)$

Pr(Worst Case) = \[
\frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{2} \approx \frac{2^n}{n!}\]
**Randomized Selection**

Given array A[1..n], find the **k-th smallest element** in A

**A Divide and Conquer Algorithm:**

Select(A, k)
1. Pick an item \( v \) in A
2. Let:
   - \( A_L = \) all elements in A that are < \( v \)
   - \( A_M = \) all elements in A that are = \( v \)
   - \( A_R = \) all elements in A that are > \( v \)
3. Return:
   - Select\( (A_L, k) \) if \( k \leq |A_L| \)
   - \( v \) if \( |A_L| < k \leq |A_L| + |A_M| \)
   - Select\( (A_R, k - |A_L| - |A_M|) \) otherwise

How to select \( v \)?

Pick \( v \) uniformly at random in 1..n

**Worst case:** \( v = \) smallest or largest element

Time \( T(n) = O(n) \) (for splitting) + \( T(n-1) \)

Solving the recurrence, \( T(n) = O(n^2) \)

**Best case:** \( v \) is the k-th element

\[
\Pr(\text{Worst Case}) = \frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{2} \approx \frac{2^n}{n!}
\]
Randomized Selection

Given array A[1..n], find the **k-th smallest element** in A

### A Divide and Conquer Algorithm:

**Select**(A, k)

1. Pick an item v in A
2. Let:
   - AL = all elements in A that are < v
   - AM = all elements in A that are = v
   - AR = all elements in A that are > v
3. Return:
   - Select(AL, k) if k <= |AL|
   - v if |AL| < k <= |AL| + |AM|
   - Select(AR, k - |AL| - |AM|) otherwise

How to select v?
Pick v uniformly at random in 1..n

**Worst case:** v = smallest or largest element
Time T(n) = O(n) (for splitting) + T(n-1)
Solving the recurrence, T(n) = O(n^2)

Pr(Worst Case) = \[ \frac{2}{n} \cdot \frac{2}{n - 1} \cdot \ldots \cdot \frac{2}{2} \approx \frac{2^n}{n!} \]

**Best case:** v is the k-th element
Time taken T(n) = O(n) (for splitting) + O(1)
Randomized Selection

Given array A[1..n], find the k-th smallest element in A

A Divide and Conquer Algorithm:

Select(A, k)
1. Pick an item v in A
2. Let:
   AL = all elements in A that are < v
   AM = all elements in A that are = v
   AR = all elements in A that are > v
3. Return:
   Select(AL, k) if k <= |AL|
   v if |AL| < k <= |AL| + |AM|
   Select(AR, k - |AL| - |AM|) otherwise

How to select v?
Pick v uniformly at random in 1..n

Worst case: v = smallest or largest element
Time T(n) = O(n) (for splitting) + T(n-1)
Solving the recurrence, T(n) = O(n^2)
Pr(Worst Case) = \( \frac{2}{n} \cdot \frac{2}{n-1} \cdot \cdots \frac{2}{2} \approx \frac{2^n}{n!} \)

Best case: v is the k-th element
Time taken T(n) = O(n) (for splitting) + O(1)
T(n) = O(n)
Randomized Selection

Given array A[1..n], find the **k-th smallest element** in A

**A Divide and Conquer Algorithm:**

`Select(A, k)`

1. Pick an item v in A
2. Let:
   - `A_L` = all elements in A that are < v
   - `A_M` = all elements in A that are = v
   - `A_R` = all elements in A that are > v
3. Return:
   - `Select(A_L, k)` if k <= |`A_L`|
   - v if |`A_L`| < k <= |`A_L`| + |`A_M`|
   - `Select(A_R, k - |A_L| - |A_M|)` otherwise

**Worst case:** v = smallest or largest element
Time T(n) = O(n) (for splitting) + T(n-1)
Solving the recurrence, T(n) = O(n^2)
Pr(Worst Case) = $\frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{2} \approx \frac{2^n}{n!}$

**Best case:** v is the k-th element
Time taken T(n) = O(n) (for splitting) + O(1)
T(n) = O(n)
Pr(Best Case) >= 1/n

How to select v?
Pick v uniformly at random in 1..n

Randomized Selection
Randomized Selection

Given array $A[1..n]$, find the **$k$-th smallest element** in $A$

### A Divide and Conquer Algorithm:

Select($A$, $k$)

1. Pick an item $v$ in $A$
2. Let:
   - $A_L = \text{all elements in } A \text{ that are } < v$
   - $A_M = \text{all elements in } A \text{ that are } = v$
   - $A_R = \text{all elements in } A \text{ that are } > v$
3. Return:
   - Select($A_L$, $k$) if $k \leq |A_L|$
   - $v$ if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R$, $k - |A_L| - |A_M|$) otherwise

**How to select $v$?**

Pick $v$ uniformly at random in $1..n$

**Average case:** Let $T(n)$ be the expected running time on an array of size $n$
Given array $A[1..n]$, find the **k-th smallest element** in $A$

**A Divide and Conquer Algorithm:**

Select($A$, $k$)

1. Pick an item $v$ in $A$
2. Let:
   - $A_L =$ all elements in $A$ that are $< v$
   - $A_M =$ all elements in $A$ that are $= v$
   - $A_R =$ all elements in $A$ that are $> v$
3. Return:
   - Select($A_L$, $k$) if $k \leq |A_L|$
   - $v$ if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R$, $k - |A_L| - |A_M|$) otherwise

**How to select $v$?**

Pick $v$ uniformly at random in $1..n$

**Average case:** Let $T(n)$ be the expected running time on an array of size $n$

**Lucky split:** $v$ is the $m$-th smallest element, for $n/4 \leq m \leq 3n/4$. 

Randomized Selection

Given array $A[1..n]$, find the **k-th smallest element** in $A$

**A Divide and Conquer Algorithm:**

Select($A$, $k$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L = \text{all elements in } A \text{ that are } < v$
   - $A_M = \text{all elements in } A \text{ that are } = v$
   - $A_R = \text{all elements in } A \text{ that are } > v$
3. Return:
   - Select($A_L$, $k$) if $k \leq |A_L|$
   - $v$ if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R$, $k - |A_L| - |A_M|$) otherwise

**Average case:** Let $T(n)$ be the expected running time on an array of size $n$

**Lucky split:** $v$ is the $m$-th smallest element, for $n/4 \leq m \leq 3n/4$. Pr[Lucky Split] = 1/2

**How to select $v$?**

Pick $v$ uniformly at random in $1..n$
Given array A[1..n], find the \textbf{k-th smallest element} in A

\textbf{A Divide and Conquer Algorithm:}

Select(A, k)
1. Pick an item v in A
2. Let:
   \[ A_L = \text{all elements in A that are } < v \]
   \[ A_M = \text{all elements in A that are } = v \]
   \[ A_R = \text{all elements in A that are } > v \]
3. Return:
   \[ \text{Select}(A_L, k) \quad \text{if } k \leq |A_L| \]
   \[ v \quad \text{if } |A_L| < k \leq |A_L| + |A_M| \]
   \[ \text{Select}(A_R, k - |A_L| - |A_M|) \quad \text{otherwise} \]

\textbf{Average case:} Let T(n) be the expected running time on an array of size n

\textbf{Lucky split:} v is the m-th smallest element, for n/4 \leq m \leq 3n/4. \quad \text{Pr}[\text{Lucky Split}] = 1/2

\[ T(n) \leq \text{Time to split} + \text{Pr}[\text{Lucky Split}] \times T(\text{array of size } \leq 3n/4) \]
\[ + \text{Pr}[\text{Unlucky Split}] \times T(\text{array of size } \leq n) \]
Given array A[1..n], find the **k-th smallest element** in A

### A Divide and Conquer Algorithm:

**Select(A, k)**

1. Pick an item v in A
2. Let:
   - A_L = all elements in A that are < v
   - A_M = all elements in A that are = v
   - A_R = all elements in A that are > v
3. Return:
   - Select(A_L, k) if k <= |A_L|
   - v if |A_L| < k <= |A_L| + |A_M|
   - Select(A_R, k - |A_L| - |A_M|) otherwise

**Average case:** Let T(n) be the expected running time on an array of size n

**Lucky split:** v is the m-th smallest element, for n/4 <= m <= 3n/4. Pr[Lucky Split] = 1/2

\[
T(n) \leq \text{Time to split} + \Pr[\text{Lucky Split}] \times T(\text{array of size } \leq 3n/4) \\
+ \Pr[\text{Unlucky Split}] \times T(\text{array of size } \leq n) \\
\leq n + (1/2) T(3n/4) + (1/2) T(n)
\]
Randomized Selection

Given array $A[1..n]$, find the $k$-th smallest element in $A$

A Divide and Conquer Algorithm:

Select($A, k$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L =$ all elements in $A$ that are $< v$
   - $A_M =$ all elements in $A$ that are $= v$
   - $A_R =$ all elements in $A$ that are $> v$
3. Return:
   - Select($A_L, k$)                     if $k \leq |A_L|$
   - $v$                                     if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R, k - |A_L| - |A_M|$)  otherwise

How to select $v$?
Pick $v$ uniformly at random in 1..n

Average case: Let $T(n)$ be the expected running time on an array of size $n$

Lucky split: $v$ is the $m$-th smallest element, for $n/4 \leq m \leq 3n/4$. $\text{Pr}[\text{Lucky Split}] = 1/2$

$$T(n) \leq \text{Time to split} + \text{Pr}[\text{Lucky Split}] \times T(\text{array of size } \leq 3n/4)$$
$$+ \text{Pr}[\text{Unlucky Split}] \times T(\text{array of size } \leq n)$$
$$\leq n + (1/2) T(3n/4) + (1/2) T(n)$$

Solving, $T(n) \leq T(3n/4) + 2n = O(n)$
Randomized Sorting

Given array $A[1..n]$, sort $A$

**QuickSort:**

Sort($A$)

1. Pick an item $v$ in $A$
2. Let:
   - $A_L = $ all elements in $A$ that are $< v$
   - $A_M = $ all elements in $A$ that are $= v$
   - $A_R = $ all elements in $A$ that are $> v$
3. Return:
Randomized Sorting

Given array A[1..n], sort A

**QuickSort:**
Sort(A)
1. Pick an item \( v \) in A
2. Let:
   - \( A_L \) = all elements in A that are < \( v \)
   - \( A_M \) = all elements in A that are = \( v \)
   - \( A_R \) = all elements in A that are > \( v \)
3. Return:
   - \( \text{Sort}(A_L) + A_M + \text{Sort}(A_R) \)
Randomized Sorting

Given array A[1..n], sort A

**QuickSort:**
Sort(A)
1. Pick an item v in A
2. Let:
   - \( A_L \) = all elements in A that are < v
   - \( A_M \) = all elements in A that are = v
   - \( A_R \) = all elements in A that are > v
3. Return:
   - Sort(\( A_L \)) + \( A_M \) + Sort(\( A_R \))

How to select v?
Pick v uniformly at random in 1..n