Randomized Algorithms and Probabilistic analysis
2-SAT Algorithm
Quicksort
Contention Resolution Protocol
Verifying Matrix Multiplication
Probability and Algorithms

Classical Algorithm Analysis
- **Algorithm**: deterministic sequence of operations to produce a unique output for any valid input
- **Complexity analysis**: worst case number of operations over all possible inputs of a given size

Randomized Algorithms
- Pseudo-random numbers are used in algorithm operations
- **Las Vegas**: gives correct result with random run-time
- **Monte Carlo**: distribution on the "correctness" of the result

Probabilistic Algorithm Analysis
- Given a random input, what is the expected running time?
- Given a random input, what is the CDF of running times?
- Input distribution could be uniform, or informed by application
Randomized Algorithms and Probabilistic analysis
2-SAT Algorithm
Quicksort
Contention Resolution Protocol
Verifying Matrix Multiplication
2-SAT Algorithm

Boolean formula in Conjunctive Normal Form (CNF):

\[ F = (X \lor \bar{W} \lor Y) \land (X \lor \bar{Y}) \land (W \lor V) \ldots \]

Variables get values in \{True, False\}

**Problem:** Given a formula with up to two variables per clause, find a Boolean assignment that satisfies all clauses.
Algorithm:
1. Start with an arbitrary assignment.
2. Repeat till all clauses are satisfied:
   1. Pick an unsatisfied clause.
   2. If the clause has one variable, change the value of that variable.
   3. If the clause has two variables, choose one uniformly at random and change its value.

What is the expected run-time of this algorithm?
**Theorem:** Assuming that the formula has a satisfying assignment, the expected run-time to find that assignment is $O(n^2)$.

- WLOG assume that all clauses have two *variables*.
- Fix one satisfying assignment.
- Let $n$ be the number of variables in the formula.
- $X_j =$ number of variables that match the satisfying assignment at time $j$
- $D_t =$ the expected number of steps to termination when $t$ variables match the satisfying assignment.
Analysis of the 2-SAT Algorithm

\( X_j = \) number of variables that match the satisfying assignment at time \( j \).

**Algorithm:** Pick an unsatisfied clause and change the value of one its two variables, chosen at random.

\[
\Pr(X_j = 1 \mid X_{j-1} = 0) = 1
\]

\[
\Pr(X_j = t \mid X_{j-1} = t-1) \geq 1/2 \quad 1 \leq j \leq n - 1
\]

**WLOG** we assume \( \Pr(X_j = t \mid X_{j-1} = t-1) = 1/2 \)
Analysis of the 2-SAT Algorithm

\( D_t = \) the expected number of steps to termination when \( t \) variables matched the satisfying assignment.

\( D_n = 0 \)

\( D_t = 1 + \frac{1}{2} (D_{t+1} + D_{t-1}) \quad 1 \leq t \leq n - 1 \)

\( D_0 = 1 + D_1 \)
Analysis of the 2-SAT Algorithm

\( D_t = \) the expected number of steps to termination when \( t \) variables matched the satisfying assignment.

\( D_n = 0 \)

\( D_t = 1 + \frac{1}{2} (D_{t+1} + D_{t-1}) \quad 1 \leq t \leq n - 1 \)

\( D_0 = 1 + D_1 \)

Guess and verify: \( D_t = n^2 - t^2 \)

**Theorem:** Assuming that the formula has a satisfying assignment, the expected run-time to find that assignment is \( \leq E[D_0] = O(n^2) \).
A Markov Chain 2-SAT Algorithm

Algorithm A:
1. Start with an arbitrary assignment.
2. Repeat until all clauses are satisfied, for up to \(2n^2\) steps:
   1. Pick an unsatisfied clause.
   2. If the clause has one variable, change the value of that variable.
   3. If the clause has two variables, choose one uniformly at random and change its value.

Theorem: Assuming that the formula has a satisfying assignment, with probability \(\geq \frac{1}{2}\) Algorithm A finds a satisfying assignment.

Proof: \(P(D_0 \geq 2E[D_0]) \leq 1/2.\)
Theorem: There is a one-sides error randomized algorithm for the 2-SAT problem that terminates in $O(n^2 \log(n))$ time, and with high probability returns an assignment when the formula is satisfiable, and always returns “UNSATISFIABLE” when no assignment exists.

Proof: Repeat Algorithm A until it finds a satisfying assignment, or up to log$(n)$ times. In each repeat, run the algorithm for $2n^2$ steps. If no assignment is found, return UNSATISFIABLE.

The probability that the algorithm does not find an assignment, when one exists, in log$(n)$ rounds is bounded by $\frac{1}{n}$.

The algorithm will not return an assignment that is not correct.
Randomized Algorithms and Probabilistic analysis

2-SAT Algorithm

Quicksort

Contention Resolution Protocol

Verifying Matrix Multiplication
A Randomized Quicksort

Procedure $Q_S(S)$;

**Input:** A set $S$.

**Output:** The set $S$ in sorted order.

1. Choose a random element $y$ uniformly from $S$.
2. Compare all elements of $S$ to $y$. Let

   \[ S_1 = \{ x \in S - \{ y \} | x \leq y \}, \quad S_2 = \{ x \in S - \{ y \} | x > y \}. \]

3. Return the list:

   \[ Q_S(S_1), y, Q_S(S_2). \]
Let $s_1, \ldots, s_n$ be the elements of $S$ is sorted order.
For $i = 1, \ldots, n$, and $j > i$, define 0-1 random variable $X_{i,j}$, s.t.
$X_{i,j} = 1$ iff $s_i$ is compared to $s_j$ in the run of the algorithm, else
$X_{i,j} = 0$.
The number of comparisons in running the algorithm is

$$T = \sum_{i=1}^{n} \sum_{j>i} X_{i,j}.$$ 

We are interested in

$$E[T] = \sum_{j>i} E[X_{i,j}] = \sum_{j>i} Pr(X_{i,j} = 1).$$

**Theorem**

$$E[T] = O(n \log n).$$
What is the Expected Running Time?

Let \( s_1, \ldots, s_n \) be the elements of \( S \) is sorted order. For \( i = 1, \ldots, n \), and \( j > i \), define 0-1 random variable \( X_{i,j} \), s.t. \( X_{i,j} = 1 \) iff \( s_i \) is compared to \( s_j \) in the run of the algorithm, else \( X_{i,j} = 0 \).

The number of comparisons in running the algorithm is

\[
T = \sum_{i=1}^{n} \sum_{j>i} X_{i,j}.
\]

We are interested in \( E[T] \).
What is the Expected Running Time?

What is the probability that $X_{i,j} = 1$?

$s_i$ is compared to $s_j$ iff either $s_i$ or $s_j$ is chosen as a “split item” before any of the $j - i - 1$ elements between $s_i$ and $s_j$ are chosen. Elements are chosen uniformly at random → elements in the set $[s_i, s_{i+1}, \ldots, s_j]$ are chosen uniformly at random.

$$E[X_{i,j}] = Pr(X_{i,j} = 1) = \frac{2}{j - i + 1}.$$  

$$E[T] = E\left[ \sum_{i=1}^{n} \sum_{j>i} X_{i,j} \right] = \sum_{i=1}^{n} \sum_{j>i} E[X_{i,j}] = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} \leq n \sum_{k=1}^{n} \frac{2}{k} = 2nH_n \leq 2n + 2n \cdot \ln(n)$$  

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$
Procedure DQ\_S(S);  
**Input:** A set $S$.  
**Output:** The set $S$ in sorted order.

1. Let $y$ be the first element in $S$.  
2. Compare all elements of $S$ to $y$. Let  
   \[
   S_1 = \{ x \in S - \{y\} \mid x \leq y \}, \quad S_2 = \{ x \in S - \{y\} \mid x > y \}. 
   \]
   (Elements is $S_1$ and $S_2$ are in the same order as in $S$.)  
3. Return the list:  
   \[
   DQ\_S(S_1), y, DQ\_S(S_2). 
   \]
Theorem

The expected run time of DQ_S on a random input, uniformly chosen from all possible permutations of S is $O(n \log n)$.

Proof.

Set $X_{i,j}$ as before. If all permutations have equal probability, all permutations of $S_i, ..., S_j$ have equal probability, thus

$$Pr(X_{i,j}) = \frac{2}{j - i + 1}.$$

$$E[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}] = O(n \log n).$$
Assumption: We have a source of independent random variables which follow a continuous uniform distribution on [0,1]: $U_1, U_2, U_3, \ldots$

In Matlab, the **rand** function provides this

Coming later: Methods for generating uniform variables…
Discrete Random Number Generation

- Input: Independent uniform variables $U_1, U_2, U_3, \ldots$

- We can use these to exactly sample from any discrete distribution using the cumulative distribution function:
  \[
  F_X(x) = P(X \leq x) = \sum_{k \leq x} p_X(k)
  \]

- Define the "inverse" of this discrete CDF:
  \[
  h(u) = \min\{x \mid F_X(x) \geq u\}
  \]
  \[
  X_i = h(U_i)
  \]

  \[
  P(X_i = k) = P(F_X(k - 1) < U_i \leq F_X(k)) = F_X(k) - F_X(k - 1) = p_X(k)
  \]

  *This function transforms uniform variables to our target distribution!*
Continuous Random Number Generation

- Input: Independent uniform variables $U_1, U_2, U_3, \ldots$
- We can use these to exactly sample from any continuous distribution using the cumulative distribution function:

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(z) \, dz$$

- Assuming continuous CDF is invertible:

$$h(u) = F_X^{-1}(u)$$

$$X_i = h(U_i)$$

$$P(X_i \leq x) = P(h(U_i) \leq x) = P(U_i \leq F_X(x)) = F_X(x)$$

This function transforms uniform variables to our target distribution!
Uniform Random Number Generation

- Assumption: We have a source of independent random variables which follow a continuous uniform distribution on \([0,1]\): \( \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \ldots \)
- In Matlab, the `rand` function provides this
- Chaotic dynamical systems are used to generate sequences of pseudo-random numbers whose distribution is approximately uniform on \([0,1]\)
- Simplest examples are linear congruential generators, but try to use more sophisticated methods!

\[ u_{i+1} = (a \bar{u}_i + c) \mod m \quad u_i = \frac{1}{m} \bar{u}_i \]

\(c\) and \(m\) should be relatively prime and large (plus more).
Examples of Uniform Generators

Linear Congruential Generator

\[ \bar{u}_{i+1} = (a \bar{u}_i + c) \mod m \]

The seed determines starting point.

RANDU: A catastrophically bad random number generator that was fairly widely used in the 1970’s

\[ \bar{u}_{i+1} = 65539 \cdot \bar{u}_i \mod 2^{31} \]

\[ u_i = 2^{-31} \bar{u}_i \]

- Constants picked for ease of hardware implementation, BUT
- Strong correlations among triples of values

Wikipedia
Examples of Uniform Generators

Mersenne Twister: Used by Matlab `rand`, period of $2^{19937} - 1$

Pseudo-random sequence looks nearly random.

Don’t try this at home!
Randomized Algorithms and Probabilistic analysis
2-SAT Algorithm
Quicksort
Contention Resolution Protocol
Verifying Matrix Multiplication
Symmetry Breaking

Ethernet Communication:

- A message is received by all nodes.
- Only one message per step.
- If two messages are sent - no message is received.
- Nodes detect collision
- No central scheduling protocol.
Contestion Resolution Protocol

- Assume $n$ nodes on the Ethernet
- In each step that a node has a message in its queue it tries to send it with probability $\frac{1}{n}$ (independent of other nodes).
- Probability of a collision in each step is $(1 - \frac{1}{n})^{n-1} \approx e^{-1}$
- Expected number of steps until a node successfully sends its message is $O(n)$

Equivalently: each node waits a number of steps distributed $U[0,n-1]$.

Very wasteful if only a few nodes have messages to send!
**Bakeoff Protocol**

**Each node protocol:**
1. $i \leftarrow 0$
2. While queue is not empty
   1. Wait a random number of steps in $[1,\ldots,2^i]$
   2. Try to send a message
   3. If a collusion detected $i \leftarrow i + 1$

**Analysis:**
Assume that $m$ nodes have non-empty queues.
After $\log m$ steps all nodes with non-empty queues have $i = \log m$
The expected wait of a message at the head of each queue in $O(m)$

The bakeoff protocol adapts to the actual load on the channel.
Randomized Algorithms and Probabilistic analysis
2-SAT Algorithm
Quicksort
Contention Resolution Protocol
Verifying Matrix Multiplication
Verifying Matrix Multiplication

Given three $n \times n$ matrices $A$, $B$, and $C$ in a Boolean field, we want to verify $AB = C$.

**Standard method:** Matrix multiplication - takes $\Theta(n^3)$ ($\Theta(n^{2.37})$) operations.

**Randomized algorithm:**

1. Chooses a random vector $\bar{r} = (r_1, r_2, \ldots, r_n) \in \{0, 1\}^n$.
2. Computes $B\bar{r}$;
3. Computes $A(B\bar{r})$;
4. Computes $C\bar{r}$;
5. If $A(B\bar{r}) \neq C\bar{r}$ return $AB \neq C$, else return $AB = C$.

The algorithm takes $\Theta(n^2)$ time.
Verifying Matrix Multiplication

Randomized algorithm:

1. Chooses a random vector \( \vec{r} = (r_1, r_2, \ldots, r_n) \in \{0, 1\}^n \).
2. Computes \( B\vec{r} \);
3. Computes \( A(B\vec{r}) \);
4. Computes \( C\vec{r} \);
5. If \( A(B\vec{r}) \neq C\vec{r} \) return \( AB \neq C \), else return \( AB = C \).

The algorithm takes \( \Theta(n^2) \) time.

**Theorem**

If \( AB \neq C \), and \( \vec{r} \) is chosen uniformly at random from \( \{0, 1\}^n \), then

\[
\Pr(AB\vec{r} = C\vec{r}) \leq \frac{1}{2}.
\]
Verifying Matrix Multiplication

Theorem

If $AB \neq C$, and $\bar{r}$ is chosen uniformly at random from $\{0,1\}^n$, then

\[
\Pr(AB\bar{r} = C\bar{r}) \leq \frac{1}{2}.
\]

Let $D = AB - C \neq 0$.

$AB\bar{r} = C\bar{r}$ implies that $D\bar{r} = 0$.

Since $D \neq 0$ it has some non-zero entry; assume $d_{11}$.

For $D\bar{r} = 0$, it must be the case that

\[
\sum_{j=1}^{n} d_{1j}r_j = 0,
\]

or equivalently

\[
r_1 = -\frac{\sum_{j=2}^{n} d_{1j}r_j}{d_{11}}.
\]
Verifying Matrix Multiplication

Theorem

If $AB \neq C$, and $\bar{r}$ is chosen uniformly at random from $\{0,1\}^n$, then

$$\Pr(AB\bar{r} = C\bar{r}) \leq \frac{1}{2}.$$ 

Let $D = AB - C \neq 0$.

$AB\bar{r} = C\bar{r}$ implies that $D\bar{r} = 0$.

Since $D \neq 0$ it has some non-zero entry; assume $d_{11}$.

For $D\bar{r} = 0$, it must be the case that

$$\sum_{j=1}^n d_{1j}r_j = 0,$$

or equivalently

$$r_1 = -\frac{\sum_{j=2}^n d_{1j}r_j}{d_{11}}.$$ 

For fixed $r_2, \ldots, r_n$ there is only one value of $r_1$ that satisfies the equality.