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?
Analysis of the 2-SAT Algorithm

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

- W.l.o.g. assume that all clause have two variables.
- Fix one satisfying assignment.
- Let $n$ be the number of variable 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 of its two variables, chosen at random.

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

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

W.l.o.g. 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

\[ D_t = \text{the expected number of steps to termination when } t \text{ variables matched the satisfying assignment.} \]

\[ D_n = 0 \]

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

\[ D_0 = 1 + D_1 \]

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 \( O(n^2) \).
Analysis of the 2-SAT Algorithm

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

**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 the algorithm until finds a satisfying assignment or up to log $n$ times. In each repeat, run the algorithm $2n^2$ steps. The probability that the algorithm does not find an assignment, when exists, in one round of $2n^2$ steps is bounded by $\frac{1}{2}$. The probability that the algorithm does not find an assignment, when exists, in log $n$ one round is bounded by $1/n$. 
Classical Algorithm Analysis

- *Algorithm*: A function that uses a deterministic sequence of operations to produce a unique output for any valid input
- *Complexity analysis*: determine the largest number of operations over all possible inputs of some size (worst case)

Randomized Algorithms

- Pseudo-random numbers are used in algorithm operations
- *Las Vegas algorithm*: Gives correct result with random run-time
- *Monte Carlo algorithm*: distribution on the correction 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
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

   $$S_1 = \{x \in S - \{y\} \mid x \leq y\}, \quad S_2 = \{x \in S - \{y\} \mid x > y\}.$$

3. Return the list:

   $$Q_S(S_1), y, Q_S(S_2).$$
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 

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

**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 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.

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

\[
E[T] = E[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}] = \sum_{i=1}^{n} \sum_{j>i} E[X_{i,j}] = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} \leq \sum_{k=1}^{n} \frac{2}{k} \leq 2nH_n = n \log n + O(n).
\]

\[
H_n = \sum_{k=1}^{n} \frac{1}{k}
\]
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 in \( S_1 \) 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 \textit{DQS} on a random input, uniformly chosen from all possible permutation of \textit{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

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

\[
E\left[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}\right] = O(n \log n).
\]
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.
Contention Resolution Protocol

• Assume $n$ nodes on the Ethernet
• In each step that a node has a message in its queue it ties 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 till a node successfully sends its message $O(n)$

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 optimize to the actual load on the channel.
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. Compute $B\bar{r}$;
3. Compute $A(B\bar{r})$;
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.
Randomized algorithm:

1. Chooses a random vector \( \vec{r} = (r_1, r_2, \ldots, r_n) \in \{0, 1\}^n \).
2. Compute \( B\vec{r} \);
3. Compute \( A(B\vec{r}) \);
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

Randomized algorithm:

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

The algorithm takes \( \mathcal{O}(n^2) \) time.

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.