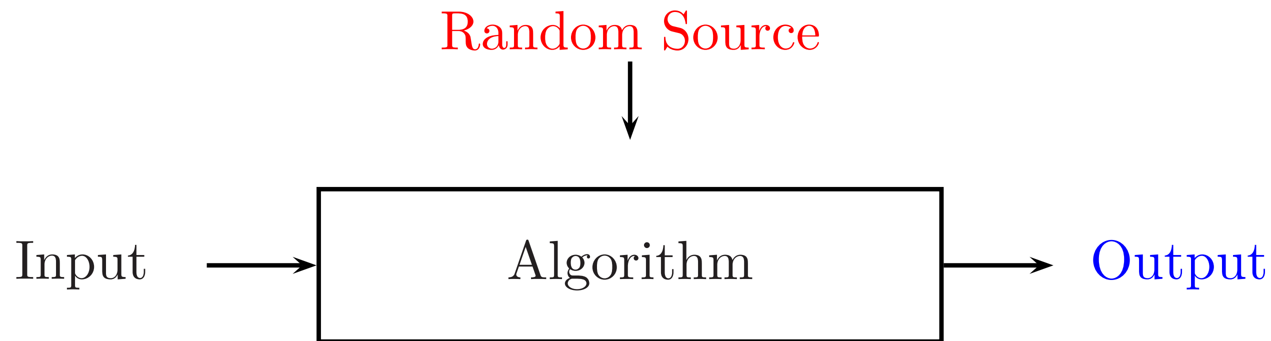
# Deterministic Algorithms

To prove that the algorithm always solves the problem as quickly as possible (in a polynomial number of steps)

# Probabilistic Analysis of Deterministic Algorithms

- The input is assumed to be from a probability distribution.
- The expected performace of the algorithm is nice
- With high probability the algorithm performs well.

# Randomized Algorithms




In adition to the input, the algorithm takes a source of random numbers and makes random choices during the execution.

The running time of a randomized algorithm may vary from run to run.

Given a problem, we wish to design algorithms + analysis to show that the behaviour of the algorithm is likely to be good, on every input of the problem

# Probabilistic Algorithms: Monte Carlo and Las Vegas

- A Monte Carlo algorithm runs for a fixed number of steps and produces an answer that is correct with probability $\geq 1/2$. However, we may make the probability of error as small as we wish amplification.
- A Las Vegas algorithm allways produces the correct answer; its running time is a random variable whose expectation is bounded (by a polynomial).

- A Las vegas algorithm can be made Monte Carlo by having it terminate with an arbitrary wrong answer, if it exceeds a time bound $t(n)$. Since Las vegas is unlikely to exceed its time bound, the resulting Monte Carlo is unlikely to give the wrong answer.
- There is not known universal method for making a Monte Carlo algorithm into a Las Vegas one.

# Random Selection

The first randomization technique we will see is Random Selection:

The intuition behind this idea is that a single randomly selected individual is probably a typical representative of the entire population.

Therefore, random selection provides a good way to avoid selecting rare bad elements.

# Verifying Polynomial Identities

Given polynomials $P(x)$ and $Q(x)$ we wish to verify if $P(x) \equiv Q(x)$

Recall: $P(x) \equiv Q(x)$ iff at every point $x$, $P(x) = Q(x)$.

**Example** Check if $(x+1)(x-2)(x+3)(x-4)(x+5)(x-6) \equiv x^6 - 7x^3 + 25$

Determinitic solution:

If $P$ and $Q$ are written explicitly, there is a linear algorithm (comparing the coefficients):

Transform $P(x)$ to canonical form $H(x) \equiv \sum_{i=0}^{d} c_i x_i$ where $d$ is the degree of $P(x)$ or of $Q(x)$.

then $P(x) \equiv Q(x)$ iff coefficients of all monomial are equal.

But for example, $Q$ can be given:

$$Q(x) = \prod_{i<j|i,j\neq 1} (x_i - x_j) - \prod_{i<j|i,j\neq 2} (x_i - x_j) + \prod_{i<j|i,j\neq 3} (x_i - x_j) - \dots$$
$$\pm \prod_{i<j|i,j\neq n} (x_i - x_j)$$

## Randomized solution

Given $P(x)$ and $Q(x)$ of degree $d$:

Schwartz-Zippel's Algorithm

**Choose** a random integer $r \in S = [0, \cdots, 100d]$
**Compute** $P(r)$ and $Q(r)$
**if** $P(r) = Q(r)$ **then** TRUE
    **else** FALSE

For example, if $r = 2$ then $P(2) = 420$ and $Q(2) = 33$, therefore as $P(2) \neq Q(x)$ the algorithm will return FALSE.

Notice that if $P(x) \equiv Q(x)$ any value of $r$ will yield equality

For example $(x+1)(x-1) \equiv x^2 - 1$.

In the case $P(x) \not\equiv Q(X)$: **A bad choice of $r$ may lead to a wrong answer**

For ex. to check if $x^2 + 7x + 1 = (x+2)^2$

If $r = 2$, $19 \neq 16$ so the two polynomials are different,

but $r = 1$, $9 = 9$ !!

Theorem: Let $F(x) = P(x) - Q(x)$ have degree $\leq d$. If $F \not\equiv 0$ then

$$\mathbf{Pr}\left[F(r) = 0\right] \leq \frac{d}{|S|} = 1\%$$

A simple event: a choice of $r$; A sample space: all $[100d]$

The probability of a simple event: $\mathbf{Pr}\left[r\right] = \frac{1}{100d}$.

By the Fundamental Theorem of Algebra, a polynomial of degree $d$ have at most $d$ roots.

Therefore: a bad event is choosing a root of $F(X)$. There as there are at most $d$ roots, then

$$\mathbf{Pr}\left[\text{bad event}\right] \leq \frac{d}{100d}.$$

- If the identity is correct, the algorithm always output the correct answer,
- If the identity is NOT correct, the algorithm outputs the wrong answer only if $r$ is a root of $F(x) = P(x) - Q(x) = 0$

# Amplification

We can decrease the error probability at the expense of increasing the run-time of the algorithm:

**Run** the algorithm $k$ times
**if** each time is TRUE **output** TRUE
    **else** FALSE

The probability of a wrong answer in one run of the algorithm is 1/100. Runs of the algorithms are independent. It is enough to get a correct (negative) answer in one of the runs.

The probablility of error in $k$ runs is $(\frac{1}{100})^k$.

# Quicksort

Recall: Given set $A$, $|A| = n$, to sort:

**select** last key to be splitter $y$

**compare** every element in $A$ to $y$ to get:

$S = \{x \in A - \{y\} | x \leq y\}, B = \{x \in A - \{y\} | x \geq y\}$ $e$

**return** $SyB$ (apply Quicksort separately to $S$ and $B$)

How to choose the splitter $y$:

Choose the rightmost input number, in the first scanning.

7 8 2 6 5 1 3 4

2 1 3 4 5 8 7 6

2 1 3 4 5 6 8 7

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

# Randomized Quicksort

Given set $A$, $|A| = n$, to sort

Rquicksort

**select** u.a.r. $y \in A$

**compare** every element in $A$ to $y$ to get:

$S = \{x \in A - \{y\} | x \leq y\}, B = \{x \in A - \{y\} | x \geq y\}$ $e$

**return** $SyB$.

Let $T$ be the number of comparisons in a run of Rquicksort

Theorem: $\mathbf{E}[T] = O(n \log n)$.

Consider a uniform distribution of the instances for Rquicksort. Given $n$ different keys, each permutation has the same probability of appearence $\frac{1}{n!}$.

$\Omega$ will consist of all possible $n!$ permutations of $A$.

For any $w \in \Omega$ $\Pr w = \frac{1}{n!}$.

Let $a_1, a_2 \ldots, a_n$ be sorted elements of $A$.

For $i \in [n]$ and $j > i$ define the indicator variable:

$$X_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is compared to } a_j \\ 0 & \text{otherwise} \end{cases}$$

The number of comparisons in running the algorithm is

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

We wish to compute $\mathbf{E}[T]$

$a_i$ is compared to $a_j$ iff either $a_i$ or $a_j$ is choosen as splitter before any of the $j - i - 1$ elements in $[a_{i+1}, \ldots, a_{j-1}]$.

Elements are choosen u.a.r. $\Rightarrow$ elemts in $[a_i, a_{i+1}, \ldots, a_{j-1}, a_j]$ are choosen u.a.r.

$\mathbf{Pr}\left[X_{i,j} = 1\right] = \frac{2}{j-i+1} \Rightarrow \mathbf{E}\left[X_{i,j}\right] = \frac{2}{j-i+1}$.

$$\begin{aligned}\mathbf{E}\left[T\right] = \mathbf{E}\left[\sum_{i=1}^{n}\sum_{j>i} X_{i,j}\right] &= \sum_{i=1}^{n}\sum_{j>i}\mathbf{E}\left[X_{i,j}\right] \\ &= \sum_{i=1}^{n}\sum_{j>i}\frac{2}{j-i+1} \leq n\sum_{k=1}^{n}\frac{2}{k} \\ &= 2nH_n = n\log n + O(n).\end{aligned}$$

# Probabilistic Analysis of Quicksort

Theorem: The expected time of Quicksort on a random input, choosen u.a.r. from all possible permutations of $A$ is $O(n \log n)$.

*Proof.* Set the r.v. $X_{i,j}$ as before. As all permutations $A_i, \ldots, A_j$ have equal probability $\mathbf{Pr}\left[X_{i,j}\right] = \frac{2}{j-i+1} \Rightarrow \mathbf{E}\left[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}\right] = O(n \log n)$. □

# A randomized algorithm for the Minimum Cut on a graph

Let $G = (V, E)$ be an undirected graph. A cut in $G$ is a set of edges who se removal separates $G$. The MINCUT problem is to find a cut of minimal size. Given $G = (V, E)$, $|V| = n$

Karger's algorithm

**while** $G$ has more than 2 vertices **do**

  select an edge $e = \{i, j\}$ uniformly at random

  contract $e$

output the remaining edges.

Contract $E$ means that the $i$ and $j$ are merged into a single vertex, retaining all their connexions to other vertices.

Notice the above algorithm:

- It is a Monte Carlo algorithm,
- $t(n) = (n-2)O(n) = O(n^2)$.
- Always outputs a valid cut in $G$ (why?)
- Need to analyze the probability it outputs a minimum cut

Let $\mathcal{C}$ be a particular minimum cut, $|\mathcal{C}| = k$. We wish to evaluate the probability that no edge in $\mathcal{C}$ is choosen in the contraction process. Let $A_i$ be the event that $\mathcal{C}$ survives iteration $i$.

**Pr** $[A_1]$: Every $j \in V$ has degree $\geq k$ (otherwise, $i$ itself would be a min. cut), so $|E| \geq \frac{nk}{2}$ (otherwise there would be an edge with degree $< k$).

$$\therefore \mathbf{Pr}\left[A_1\right] \geq 1 - \frac{k}{(kn/2)} = 1 - \frac{2}{n} = \frac{n-2}{n}.$$

To see the prob. that $\mathcal{C}$ survives the second iteration given that it survived the first, notice at second step $|E| \geq \frac{k(n-1)}{2}$, so probability of choosing an edge from $\mathcal{C}$ at second iteration is $\leq \frac{k}{k(n-1)/2} = \frac{2}{n-1}$.

$$\therefore \mathbf{Pr}\left[A_2\right] \geq 1 - \frac{2}{n-1} = \frac{n-3}{n-1}.$$

At the $i$-th iteraction, we have $|V| = n - i + 1$, $|\mathcal{C}| = k$, therefore $|E| \geq \frac{k(n-i+1)}{2}$.

$$\therefore \mathbf{Pr}\left[A_i | \cap_{j=1}^{i-1} A_j\right] \geq 1 - \mathbf{Pr}\left[A_i\right] = 1 - \frac{k}{k(n-i+1)/2} = \frac{n-i-1}{n-i+1}$$

The probability that no edge in $\mathcal{C}$ is choosen during the whole process:

$$\mathbf{Pr}\left[\cap_{i=1}^{n-2} A_i\right] \geq \frac{n-2}{n} \times \frac{n-3}{n-1} \times \ldots \times \frac{2}{4} \times \frac{1}{3} = \frac{2}{(n-1)n}.$$

Therefore the probability the algorithm gives a mistake is less or equal than $(1 - \frac{2}{n^2})$

How to reduce more this probability?. Amplify.

Repeat independently $t$ times the algorithm, and choose the smallest cut we fi nd. The probability that we fail to discover $\mathcal{C}$ on all $t$ attempts is $\leq (1 - \frac{2}{n^2})^t$. So taking $t = cn^2$, the probability of error is at most $e^{-c/2}$. Therefore, to make the probability that the algorithm fails $= 10^{-6} = e^{-14}$ we just need to do $28n^2$ repetitions.