# Randomized Computation: Examples and Complexity Classes

## 1 Introduction

This lecture, we will study classical examples of randomized algorithms and also complexity classes introduced by randomness in computation. We will cover important developments such as primality testing, verification of algebraic identities, and study complexity classes such as RP, BPP, and ZPP and their relation to other classes such as P, NP, PSPACE, and the like. Along the way, we also study the notion of probability amplification via repeated trials.

## 2 Primality Testing

The importance of prime numbers to the field of cryptography cannot be under-emphasized. Many cryptographic protocols depend on the ability to generate arbitrarily large prime numbers. Let us start by recalling that a natural number $n$ is said to be *prime* if $n$ has no divisors except 1 and itself. The number 2 is the smallest prime and the list of primes less than 20 are 2,3,5,7,11,13,17, and 19.

In this section, we shall focus on the distribution of primes, testing primality, and generating prime numbers. In an important discovery, it was shown by whosisit that the number of primes is countably infinite and also the following theorems were shown.

**Theorem 2.1** *Between a prime number $p$ and $2p$ there exists at least one another prime number.*

The bound of $2p$ can be improved to $p + p^{2/3}$ via advanced arguments. However, to show the existence of a countably infinite set of primes the above theorem suffices. The next theorem shows that there are enough prime numbers.

**Theorem 2.2 (Prime Number Theorem)** *Let $p(n)$ denote the number of primes between the natural numbers 2 and $n$. Then,*

$$p(n) \rightarrow \frac{n}{\log n} \text{ as } n \rightarrow \infty$$

This leads us to our next question of how to recognize that a given number is prime? An efficient algorithm for this task has eluded number theorists and computer scientists alike for many decades till Agarwal, Kayal, and Saxena [**?**] have come up with a breakthrough. But for pedagogical interest we review the prominent developments in this area.

### 2.0.1 Fermat's Little Theorem

From the results of the Section **??**, recall that $Z_p^*$ is a (cyclic) group with order $p-1 = \phi(p)$. Following the fundemental theorem of groups **??** the order of any element $a \in Z_p^*$ divides $\phi(p)$ [[***check this again***]]. Hence, we arrive at the following theorem called as the Fermat's little theorem.

**Theorem 2.3 (Fermat Little Theorem)** *For any $a \in Z_p^*$, it holds that $a^{p-1} \equiv 1 \bmod p$.*

The above theorem gives us a necessary condition for testing the primality of a natural number. However, consider using the above test for $n = 561 = 3 \cdot 11 \cdot 17$ which is not prime. If we choose $a = xxx$ then we get $a^{p-1} \equiv 1 \bmod n$ but still $n$ is not prime.

Traditionally, an odd composite number $n$ such that $a^{n-1} \equiv 1(\mathrm{mod}\, n)$ is called a pseudoprime to base $a$ or simply as pseudoprimes. Some authors call it as Fermat pseudoprime or ordinary pseudoprime [**?**] and is denoted by pseudoprime($a$).

Using Fermat's little theorem one can show that a number $n$ is composite. All that is required is to find a *witness* $a$ such that $a \not\equiv 0(\mathrm{mod}\, n)$ and $a^{n-1} \not\equiv 1(\mathrm{mod}\, n)$. However, the converse is not true. If it were, the question of testing primality would be settled for ever.

## 2.1 The Solovay-Strassen Test

The next set of developments in testing primality require some more notation and definitions.

**Definition 2.4 (Residues)** *Let $m$ and $n$ be positive integers. Let $a$ be an integer relatively prime to $n$, i.e., gcd$(a, n) = 1$. Then we say that $a$ is an $m$the power residue modulo $n$ iff $x^m \equiv a(\mathrm{mod}\, n)$ is solvable for integral $x$. If $m = 2$ in the above defintion, then $a$ is called a quadratic residue. If $m = 2$ and $x^2 \equiv a(\mathrm{mod}\, n)$ is not solvable then $a$ is called a quadratic non-residue.*

Euler has shown the following important result concerning prime numbers.

**Theorem 2.5 (Euler's Prime Criterion)** *Let $n$ be an odd prime and let gcd$(a, p) = 1$. Then, $a$ is a quadratic residue modulo $p$ if and only if*

$$a^{\frac{p-1}{2}} \equiv 1(\mathrm{mod}\, p)$$

*and $a$ is a quadratic non-residue modulo $p$ if and only if*

$$a^{\frac{p-1}{2}} \equiv -1(\mathrm{mod}\, p)$$

The above resuls has a generalization. See [**?**, Theroem 5.7.2] for details.
We introduce another defnition.

**Definition 2.6 (Legendre Symbol)** *Let $a$ be an integer and $p$ be an odd prime. The Legendre symbol $\left(\frac{a}{p}\right)$ is defined as:'*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \\ 0 & \text{if } p \text{ divides } a \end{cases}$$

Thus, Euler's criterion can be rephrased as $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}}(\mathrm{mod}\, p)$. This has been used to derive a randomized algorithm for testing primality by Solovay and Strassen which we desribe now.

```
Algorithm Solovay-Strassen($n$)
  1. choose a random number uniformly in $\{1, 2, \cdots, n-1\}$
  2. if gcd($a, n$) $\neq 1$ then print "$n$ is composite"
  else
      3. if $\left(\frac{a}{p}\right) \not\equiv a^{\frac{n-1}{2}}(\mathrm{mod}\, p)$ then
          print $n$ is composite; return;
      else print $n$ is prime; return;
```

The success of the above algorithm is based on the following lemma. Let $W(n) = \{a \in Z_n^* | \left(\frac{a}{p}\right) \equiv a^{\frac{n-1}{2}}(\mathrm{mod}\, p)\}$.

**Lemma 2.7** *Let $n \geq 3$ be an odd integer. Then $n$ is prime if and only if $W(n) = Z_n^*$.*

**Proof.** If $n$ is prime, then the conclusion is true by Euler's criterion. To prove the converse, let $W(n) = Z_n^*$ but $n$ is composite. Then by the hypothesis for all $a \in Z_n^*$ it holds that:

$$a^{n-1} \equiv \left(\frac{a}{p}\right)^2 \equiv 1(\mathrm{mod}\, n)$$

If the above holds and $n$ is composite then $n$ is square-free (cf. [**?**, Theorem 9.3.6]). Let $n = pr$ where $p$ is a prime and $r > 1$ and $\gcd(p, r) = 1$. Now consider an $a$ such that $a \equiv 1 (\bmod\, r)$ and $a \equiv g (\bmod\, p)$ where $g$ is a quadratic non-residue modulo $p$. Note that such an $a$ exists by Chinese remainder theorem. Then,

$$\left(\frac{a}{n}\right) = \left(\frac{a}{pr}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{r}\right) = \left(\frac{g}{p}\right)\left(\frac{1}{r}\right) = (-1)(+1) = -1$$

□

Let us now analyze the result of the above algorithm. Notice that if $n$ is an odd prime then by the Euler's criterion that $\left(\frac{a}{p}\right) = a^{(n-1)/2} (\bmod\, n)$ for all $a \in Z_p^*$ the algorithm always prints "prime". But on the other hand, if $n$ is odd and composite and $a$ is not relatively prime to $n$ then the output of the algorithm depends on $\left(\frac{a}{n}\right)$. Using the above lemma, $W(n) \neq Z_n^*$. It is easy to check that $W(n)$ is a sub-group and by the previous lemma it is a proper subgroup. It then holds that $|W(n)| \leq |Z_n^*|/2 = (n-1)/2$ as the order of a sub-group divides the order of a group. Thus, for at least half the $a$ chosen, if $n$ were odd composite then the test prints "composite".

From the above we have:

**Theorem 2.8** *Composites* $\in RP$.

## 3 Set Balancing Problem

In this section, we consider another application of Chernoff bounds. The problem is called Set Balancing Problem and is defined as follows. Given an $n \times n$ $\{0, 1\}$ matrix $A$, find a $\{-1, +1\}$ valued column vector $X$ such that the product $AX$ has the smallest maximum absolute entry.

**Example. 3.1** *Let the matrix A be as given below.*

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

$$\textit{For } X = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, AX = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix}.$$

*Thus the maximum absolute entry is* 2.

Our goal is to make every entry of $AX$ to be as close $0$ as possible. So we are in a way measuring the discrepancy by the maximum absolute value of the entries of $AX$. In general, it might not be possible to make every entry of $AX$ to be $0$, for example if $A$ has a row with odd number of $1's$ as in the above example. A brute force solution for choosing $X$ involves trying all possible column vectors of size $n$ and this would take $\Omega(2^n)$ time. Instead, we will see that there exists a very simple randomized algorithm that guarantees an expected discrepancy of $O(\sqrt{n \ln n})$. Later on, we can derandomize our algorithm and thus obtain a deterministic polynomial time algorithm with the same guarantee on the discrepancy. It is interesting to note that for this problem, Spencer [**?**] proved that for any matrix $A$ there is a column vector $X$ such that the discrepancy is at most $6\sqrt{n}$. It is not known whether there is a polynomial time randomized or deterministic algorithm that guarantees a discrepancy of $O(\sqrt{n})$.

The randomized algorithm works as follows. Let $X = [X_1 X_2 \cdots X_n]^T$. Choose each $X_i$ independently and u.a.r with $Pr[X_i = +1] = Pr[X_i = -1] = 1/2, 1 \leq i \leq n$. Before analyzing the performance guarantee of the randomized algorithm we state the classic Boole's inequality.

**Fact 3.2 Boole's Inequality:**
*Let* $E_1, E_2, \cdots E_n$ *be* $n$ *events. Then,* $Pr[E_1 \cup E_2 \cup \cdots \cup E_n] \leq Pr[E_1] + Pr[E_2] + \cdots + Pr[E_n]$.

Boole's inequality has the following application. Let the events $E_i$ be bad events. Then the union of all these bad events defines the event that at least one bad event occurs. To be able to prove that no bad event occurs with high probability, we can bound the probability that some bad event occurs by estimating the probability of each bad event (independently even though we are not given that all bad events are independent) and summing the probabilities.

Let the product $AX = Y$. Consider any $Y_i$, say $Y_1$. By definition of matrix multiplication, $Y_1 = A_{11}X_1 + A_{12}X_2 + \cdots + A_{1n}X_n$ where the $A_{ij}$ denotes the element of $A$ at $i^{th}$ row and $j^{th}$ column. Note that $E[X_i] = 0$ and by linearity of expectations, $E[Y_1] = 0$. For $d = 8\sqrt{n \ln n}$, using Corollary **??** we get $Pr[Y_1 \geq d] \leq e^{\left(\frac{-64n \ln n}{8n}\right)} = e^{-8 \ln n} = \frac{1}{n^8}$. But we are interested in a two-sided bound. That is, since we want to minimize the absolute value of $Y_1$, we need to compute $Pr[Y_1 \leq -d]$ also. But by symmetry, $Pr[Y_1 \leq -d] = Pr[Y_1 \geq d] \leq \frac{1}{n^8}$. Thus,

$$Pr[|Y_i| \geq 8\sqrt{n \ln n}] \leq \frac{2}{n^8}, 1 \leq i \leq n. \tag{1}$$

Let us interpret each event $|Y_i| \geq 8\sqrt{n \ln n}$ as a bad event. Thus using Boole's inequality, $Pr[$ for some $i$, $|Y_i| \geq 8\sqrt{n \ln n}] \leq \sum_{i=1}^{n} Pr[|Y_i| \geq 8\sqrt{n \ln n}] = n\frac{2}{n^8} = \frac{2}{n^7}$. Thus with probability greater than $1 - \frac{2}{n^7}$, every entry in $Y$ has absolute value at most $8\sqrt{n \ln n}$. Indeed, the expected value of the maximum absolute value is at most $(1 - \frac{2}{n^7})8\sqrt{n \ln n} + \frac{2}{n^7}n$. Here we have assumed that when the absolute value is greater than $8\sqrt{n \ln n}$, $|Y|$ assumes the maximum possible value of $n$.

## 4 Fingerprinting and Freivalds Technique

It is known that randomization combined with (non-trivial) algebraic techniques can lead to important applications. In this section, we showcase some of such techniques with respect to verification of identities. One such technique is called the *fingerprinting* technique roughly described as follows. Let $U$ be any universe and $x$ and $y$ are two elements for which we ask *Is* $x = y$?. One can answer this using at least $\log |U|$ bits in a deterministic manner. However, consider mapping elements of $U$ to a sparse universe $V$ such that $x$ and $y$ are identical if and only if their images in $V$ are identical, with a good chance. These images can be thought of as fingerprints of $x$ and $y$.

Let us apply the above technique to matrix product verification. Let $\mathbb{F}$ be a field and $A$ and $B$ are two matrices with entires from $\mathbb{F}$. Suppose it is claimed that $C = A \cdot B$. The fastest known matrix multiplication algorithm runs in time $O(n^{2.376}$. This algorithm is very difficult to implement, but the standard algorithms such as Strassen's recursive algorithm takes time $O(n^{\log_2 7})$. So, to verify if $C$ is indeed the product of $A$ and $B$, it takes time equal to multiplying two matrices.

However, a simpler and efficient randomized approach exists. Let $\mathbf{r}$ be any vector with entires being 0 or 1. Let each element of $\mathbf{r}$ be chosen independently and uniformly at random. (It is being assumed without loss of generality that 0 and 1 are the additive and multiplicative identities of the field $\mathbb{F}$. Compute $\mathbf{x} = Br$, and $\mathbf{y} = A\mathbf{x}$. Similarly, compute $\mathbf{z} = C\mathbf{r}$. If $A \cdot B = C$ is indeed true, then $\mathbf{y}$ must equal $\mathbf{z}$ for any $\mathbf{r}$. Also, $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$ can each be computed in $O(n^2)$ time. So, the time efficiency is established. It remains to see the verification efficiency. The following lemma argues that the verification procedure is efficient.

**Lemma 4.1** *Let $A, B$, and $C$ be $n \times n$ matrices from $\mathbb{F}$ such that $AB \neq C$. Then, for $\mathbf{r}$ chosen uniformly at random from $\{0,1\}^n$, $\Pr(AB\mathbf{r} = C\mathbf{r}) \leq 1/2$.*

**Proof.** Consider the matrix $D := AB - C$. Since, $AB \neq C$, matrix $D$ is not the matrix of all zeros. We are interested in the event that $D\mathbf{r} = 0$. Assume without loss of generality that the first row of $D$ has a nonzero entry and all all nonzero entries in that row are before any zero entry.

Consider the first row of $D$ and the scalar obatined by multiplying the first row of $D$ with $\mathbf{r}$. The result is zero if and only if:

$$\mathbf{r}_1 = -\frac{\sum_{i=1}^{k} D_{1i}\mathbf{r}_i}{D_{11}}$$

In the above, it is assumed that there are $k > 0$ nonzero elements in the first row of $D$.

The above event is a superevent of the event that $D\mathbf{r} = 0$. Therefore the probability of the event $D\mathbf{r} = 0$ is upperbounded by the probability of the above event. To compute the above probability, imagine that all the choices $r_2, \cdots, r_k$ have been made. In that case, the right hand side is a scalar from the field $\mathbb{F}$. The left hand size is a value uniformly chosen amongst two values in $\mathbb{F}$. The required probability therefore cannot exceed $1/2$. (Note: The calculation is rather a crude estimate, but is sufficient for our purposes). $\square$

To improve the verification efficiency of the procedure, we can also use repeated independent trials. Let us perform $t$ independent trials of the above procedure. For $AB \neq C$, the probability that the test fails in each trial is at most $1/2$. So, in $t$ trials, the probability that all $t$ trials fail, or a majority of the $t$ trials fail, is at most $1/2^t$. For $t = O(\log n)$, the failure probability is polynomially small. This technique is called as probability amplification and helps boost the success probability of co-RP algorithms.

Other applications of the technique include verifiying polynomial identities. For instance, let $P_1(x), P_2(x)$ be two polynomials in a field $\mathbb{F}$. The polynomial product verification problem is to check whether $P_1(x) \cdot P_2(x) = P_3(x)$ for a given $P_3(x)$. It holds that there exists an $O(n \log n)$ time algorithm to multiply two polynomials, where $n$ is the maximum degree of $P_1$ and $P_2$. We design a verification algorithm that is faster than $O(n \log n)$.

Let $\mathbb{S} \subset \mathbb{F}$ be a susbet of size at least $2n + 1$. The main idea of the verification procedure is that if indeed $P_3(x)$ equals $P_1(x) \cdot P_2(x)$, then, also $P_3(r) = P_1(r) \cdot P_2(r)$ for $r$ chosen uniformly at random from $\mathbb{S}$. Further, evaluating a polynomial at a given input can be done in $O(n)$ time. So, we can declare that $P_3(x)$ equals $P_1(x) \cdot P_2(x)$ unless $P_3(r) \neq P_1(r) \cdot P_2(r)$. The algorithm makes a mistake only when indeed $P_3(x) \neq P_1(x) \cdot P_2(x)$ but the choice of $r$ fails to detect this.

To estimate the probability that the algorithm makes a mistake, let $Q(x) := P_3(x) - P_1(x) \cdot P_2(x)$. The degree of $Q(x)$ is at most $2n$. Suppose that $P_3(x) \neq P_1(x) \cdot P_2(x)$. Then, $Q(x)$ is a nonzero polynomial. So the test fails if $Q(r) = 0$. However, the polynomial $Q(x)$ of degree at most $2n$ can have at most $2n$ roots. So, the probability that $Q(r) = 0$ is at most $2n/|\mathbb{S}|$, which is also the probability of error. As earlier, the probability of failure can be made polynomially small in $n$ by using repeated trails or choosing a larger $\mathbb{S}$, or both.

One may wonder whether it is at all worthwhile to have elaborate verification algorithms for things as simple as polynomial product verification. Such techniques however are more applicable when polynomials are not available explicitly.

## 5 Classes RP , BPP , and ZP

The classes of complexity such as P, NP can be generalized to the case of randomized computations as follows. We start with the following defintiions.

**Definition 5.1** *The class RP consists of languages $L$ such that there exists a randomized algorithm running in worst case polynomial time such that for any input $x$:*

- $x \in L \Rightarrow \Pr(A \text{ accepts } x) \geq 1/2$.
- $x \notin L \Rightarrow \Pr(A \text{ accepts } x) = 0$.

Observe from the above definition that an RP algorithm cannot make a mistake when given negative instances. This is also called as making a *one-sided error*. Further, repeated trials can also decrease the error probability to be polynomially small. This does increase the runtime, but in most cases, by a polynomial factor.

The class co-RP can be defined as the class where error is allowed only for negative instances. The following definition can be given for co-RP :

**Definition 5.2** *The class coRP consists of languages $L$ such that there exists a randomized algorithm running in worst case polynomial time such that for any input $x$:*

- $x \in L \Rightarrow \Pr(A \text{ accepts } x) = 0$.
- $x \notin L \Rightarrow \Pr(A \text{ accepts } x) < 1/2$.

Notice that there can be algorithms with zero-sided error as seen from the following definition.

**Definition 5.3** *The class ZP consists of languages $L$ such that there is a randomized algorithm $A$ that always outputs the correct answer while running in expected polynomial time.*

It is not difficult to show that ZP $=$ RP $\cap$ co-RP . Randomized algorithms are also classfied as two kinds: Las Vegas and Monte Carlo. Las Vegas algorithms always give the correct solution, but may vary on their runtime. So, the class ZP corresponds to Las Vegas algorithms. On the other hand, Monte Carlo algorithms may sometimes provide an incorrect answer. It is possible to reduce the probability of error using repeated trials, in most cases. RP and co-RP algorithms are examples of Monte Carlo algorithms.

There is finally one more definition corresponding to the possibility of a two sided error.

**Definition 5.4** *The class PP consists of languages $L$ such that there exists a randomized algorithm running in worst case polynomial time such that for any input $x$:*

- $x \in L \Rightarrow \Pr(A \textit{ accepts } x) > 1/2$.
- $x \notin L \Rightarrow \Pr(A \textit{ accepts } x) < 1/2$.

Since the above definition allows even arbitrarily small separation of error probabilities from 1/2, it is not always the case that repeated polynomial trials can help. To resolve this issue, a more tighter definition such as the following is used.

**Definition 5.5** *The class BPP (Bounded-error Probabilistic Polynomial) consists of languages $L$ such that there exists a randomized algorithm running in worst case polynomial time such that for any input $x$:*

- $x \in L \Rightarrow \Pr(A \textit{ accepts } x) \geq 3/4$.
- $x \notin L \Rightarrow \Pr(A \textit{ accepts } x) \leq 1/4$.

In the above definition, the numbers 3/4 and 1/4 are rather arbitrary. These can be weakened to $1/2 + 1/p(n)$ and $1/2 - 1/p(n)$ for some polynomial $p(n)$. Such polynomially away error probabilities suffice to make use of repeated polynomial trails.

The following claims can be verified:

- P $\subseteq$ RP $\subseteq$ NP $\subseteq$ PSPACE $\subseteq \exp \subseteq$ NEXP .
- RP $\subseteq$ BPP $\subseteq$ PP .
- PP $=$ co-PP , and BPP $=$ co-BPP .
- NP $\subseteq$ PP $\subseteq$ PSPACE.