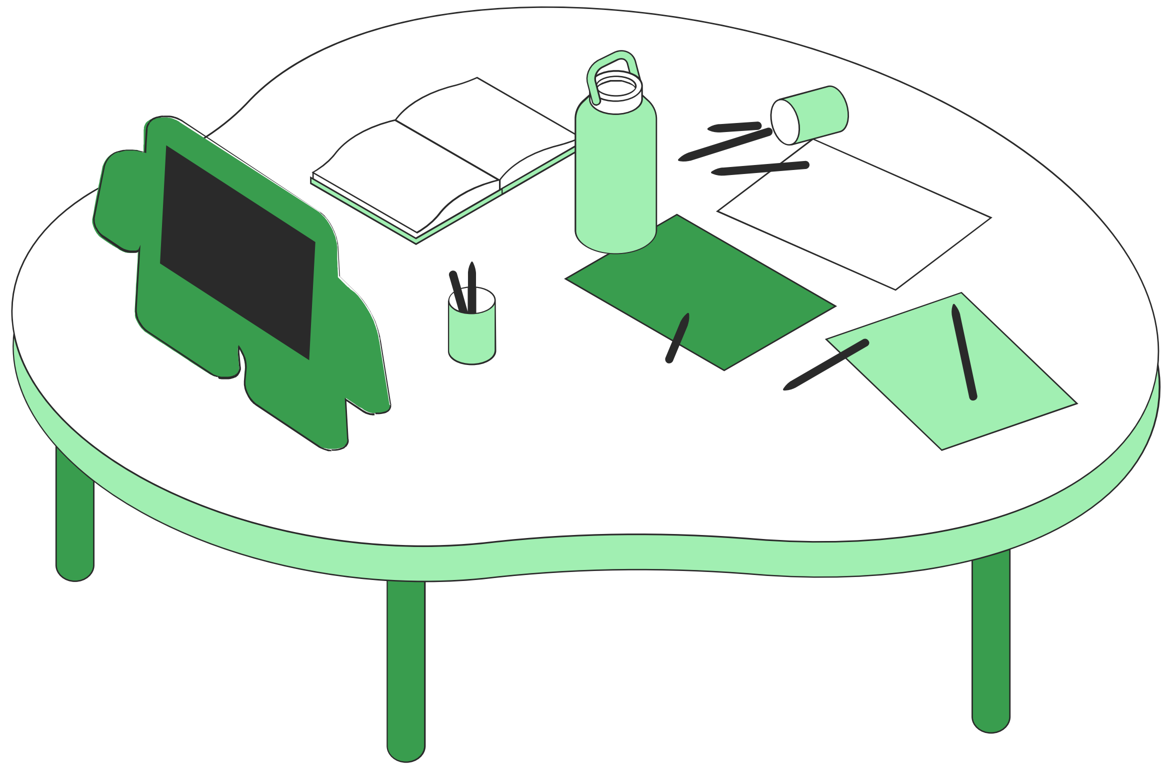
De-Randomization in Advanced Algorithms

By: Aayushi Singh & Yash Kushwaha
Roll Num: 300012721002 & 300012721040
CSE(A.I): 5th Sem

What to Know



Overview of Randomized Algorithms

Why Derandomization is Important:

Introducing Derandomized Advanced Algorithms

Deterministic approaches to replace randomization

- Randomized algorithms are algorithms that use randomness to achieve their goals. Unlike deterministic algorithms that produce the same output for a given input every time they run, randomized algorithms make use of random numbers to introduce an element of uncertainty.
- This randomness is leveraged to achieve certain advantages, such as simplicity, efficiency, and the ability to handle probabilistic problems.
- Common examples of randomized algorithms include randomized quicksort, Monte Carlo algorithms, and Las Vegas algorithms.
- Randomization can be particularly useful in situations where finding an exact solution might be computationally expensive or even impractical.

Why De-randomization is Important:

- The reliance on randomness makes these algorithms non-deterministic, which means that their behavior can vary across different runs.
- This variability may not be acceptable in certain applications where consistency and predictability are crucial.

- De-randomization addresses these concerns by providing deterministic counterparts to randomized algorithms. Derandomized algorithms aim to achieve the same or similar probabilistic guarantees as their randomized counterparts but without relying on actual randomness during execution.
- This deterministic nature makes derandomized algorithms more predictable and easier to analyze, which is often desirable in critical applications.

Introducing Derandomized Advanced Algorithms

- Derandomized advanced algorithms represent a category of algorithms that have been modified or designed to operate without relying on randomness. These algorithms use various techniques, such as pseudorandom generators, extractors, and explicit constructions, to replace or eliminate the need for random choices.
- The goal of de-randomization is to strike a balance between the advantages of randomization and the need for determinism. Derandomized advanced algorithms find applications in areas where the unpredictability of randomized algorithms is a concern, such as in security-sensitive systems, real-time applications, and scenarios where the results must be reproducible.
- In essence, derandomized advanced algorithms aim to combine the efficiency and simplicity of randomized algorithms with the predictability and determinism of deterministic algorithms, offering a versatile solution to a wide range of computational challenges.

Deterministic approaches to replace randomization

Derandomization involves replacing or eliminating the need for randomness in algorithms by using deterministic approaches.

Pseudorandom Generators (PRGs):

Definition: Pseudorandom generators are algorithms that deterministically expand a short random seed into a longer pseudorandom sequence.
Role in De-randomization: PRGs are used to simulate randomness in a deterministic manner. By providing a pseudorandom sequence, they replace the need for true randomness in algorithms.
Examples: Cryptographic hash functions used as PRGs.

Deterministic Randomized Rounding:

Definition: Deterministic randomized rounding is a technique used in optimization problems where randomization is typically applied. It involves designing deterministic procedures that mimic the probabilistic rounding process.
Role in De-randomization: By using deterministic rounding procedures, the algorithm achieves similar results to randomized rounding without relying on random choices.
Examples: Deterministic rounding in linear programming, such as the method of conditional expectations.

A pseudorandom generator (PRG)

A pseudorandom generator (PRG) is an algorithm that deterministically transforms a short, truly random seed into a longer pseudorandom sequence that appears to be random. PRGs are widely used in various applications, including cryptography, simulations, and randomized algorithms. The basic idea is to use a small amount of true randomness to generate a larger amount of pseudorandom data.
Here's a simple explanation of a basic pseudorandom generator algorithm:

Linear Congruential Generator (LCG):

The LCG is one of the simplest and oldest pseudorandom generator algorithms. It produces a sequence of numbers using a linear congruential recurrence relation. The formula for generating the next pseudorandom number.

$$X(n+1) = (a*X(n) + c) \bmod m$$

- X(n) is the current pseudorandom number,
- a is a multiplier,
- c is an increment,
- m is the modulus,
- \mod denotes the modulo operation.

$X_{n+1} = (3X_n+7) \bmod 13$

In this example:

Multiplier (a): 3

Increment (c): 7

Modulus (m): 13

Let's generate a sequence of pseudorandom numbers starting with a seed value. For simplicity, we'll start with $X_0 = 2$

$X_{1} = (3 \times 2+7) \bmod 13 = 1$

$X_{2} = (3 \times 1+7) \bmod 13 = 10$

$X_{3} = (3 * 10 + 7) \bmod 13 = 2$

$X_{4} = (3 * 2 + 7) \bmod 13 = 1$

Here, the sequence starts to repeat after a few iterations. It's important to note that the period of the sequence (the number of unique values before it repeats) is dependent on the choice of parameters. In this case, the period is relatively short.

TRY TO SOLVE THE ABOVE USING MOD 17.

.