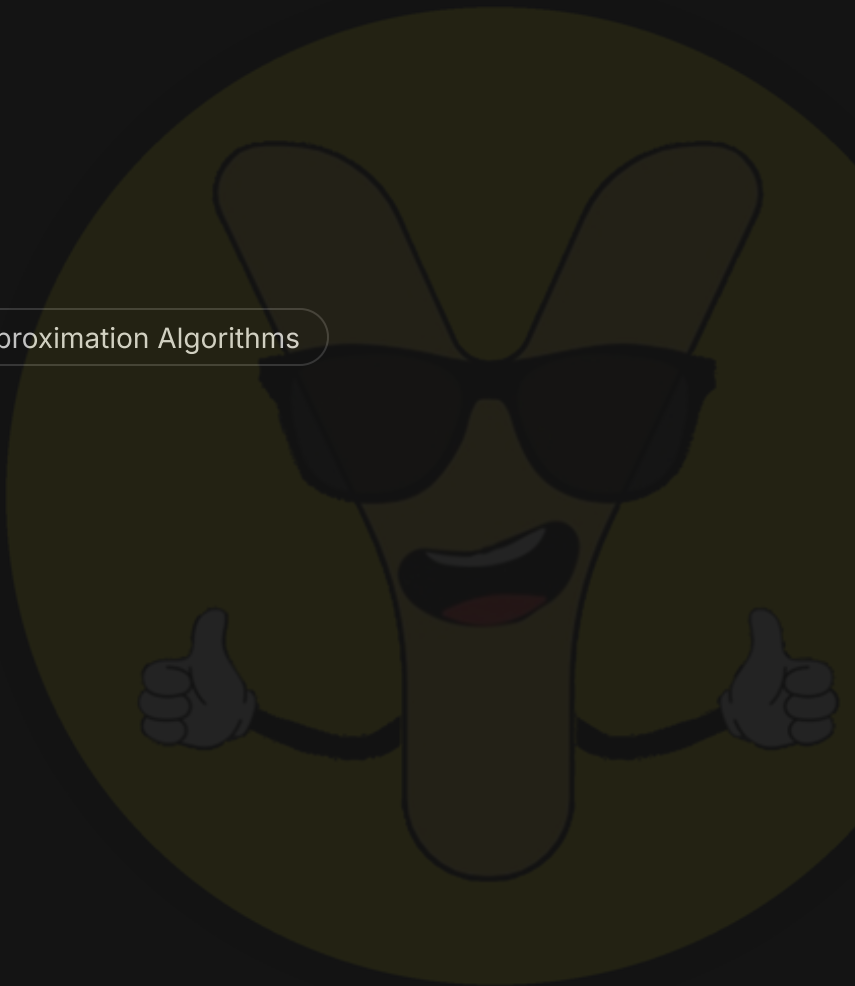
AKTU · 3RD YEAR · SEMESTER 5 · BCS503

Design and Analysis of Algorithm

Unit 5 — Selected Topics

- Algebraic Computation
- Fast Fourier Transform
- String Matching
- Theory of NP Completeness
- Approximation Algorithms
- Randomized Algorithms



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

Algebraic Computation: Overview

ALGEBRAIC COMPUTATION

Algebraic computation involves algorithms that manipulate symbolic algebraic expressions, polynomials, matrices, and finite fields, operating on exact mathematical structures rather than floating-point approximations.

- Focuses on exact algebraic operations over integers, rational numbers, and finite fields.
- Key applications include polynomial multiplication, matrix operations, and symbolic computer algebra.
- Time complexity is analyzed by counting basic algebraic operations (additions, multiplications, divisions).

Polynomial Evaluation using Horner's Rule

Evaluate the polynomial $P(x) = 3x^3 + 2x^2 - 5x + 7$ at $x = 2$ using Horner's Rule.

1. Rewrite polynomial in nested parenthesized form: $P(x) = ((3x + 2)x - 5)x + 7$
2. Initialize constant term: $b_3 = 3$
3. Compute $b_2 = b_3 \cdot x + a_2 = 3(2) + 2 = 8$
4. Compute $b_1 = b_2 \cdot x + a_1 = 8(2) - 5 = 11$
5. Compute $b_0 = b_1 \cdot x + a_0 = 11(2) + 7 = 29$

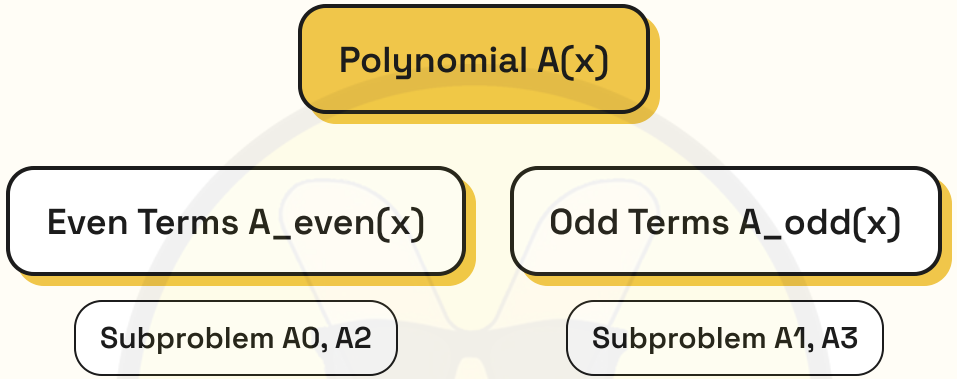
The value of $P(2)$ is 29. Horner's Rule requires only 3 multiplications and 3 additions ($O(n)$ operations vs $O(n^2)$ naive).

Discrete Fourier Transform (DFT)

DFT maps a polynomial from coefficient representation to point-value representation at complex roots of unity.

- Coefficient Form: Polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ represented by coefficients $(a_0, a_1, \ldots, a_{n-1})$.
- Evaluation Points: Evaluates polynomial at n-th complex roots of unity $\omega_n^k = e^{2\pi i k/n}$.
- DFT Formula: $y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$ for $k = 0, 1, \ldots, n-1$.
- Naive Computation: Directly evaluating DFT takes $O(n^2)$ complex arithmetic operations.
- Inverse DFT: Converts point-value form back to coefficients in $O(n \log n)$ time.

FFT Divide-and-Conquer Decomposition

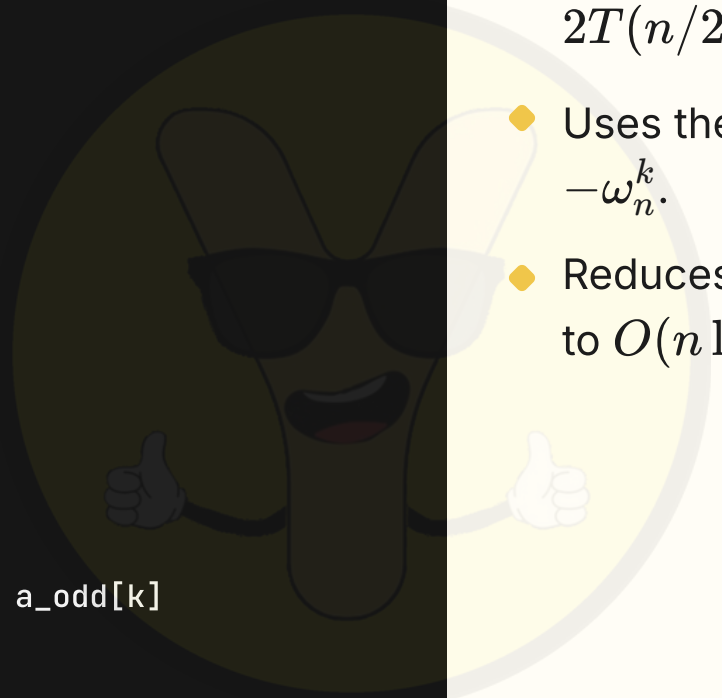


FFT splits degree n-1 polynomial into two n/2-1 degree polynomials using even and odd index coefficients.

Recursive Fast Fourier Transform (FFT)

```python
import cmath

def recursive_fft(a):
    n = len(a)
    if n == 1:
        return a
    wn = cmath.exp(2j * cmath.pi / n)
    w = 1
    a_even = recursive_fft(a[0::2])
    a_odd = recursive_fft(a[1::2])
    y = [0] * n
    for k in range(n // 2):
        y[k] = a_even[k] + w * a_odd[k]
        y[k + n // 2] = a_even[k] - w * a_odd[k]
        w = w * wn
    return y
```



- Time Complexity recurrence: $T(n) = 2T(n/2) + O(n) = O(n \log n)$.
- Uses the cancellation property: $\omega_n^{k+n/2} = -\omega_n^k$.
- Reduces polynomial multiplication from $O(n^2)$ to $O(n \log n)$.

String Matching Problem & Naive Approach

String matching finds all valid occurrences of a pattern P within text T.

- Problem Formulation: Text $T[1 \ldots n]$ and pattern $P[1 \ldots m]$ over alphabet Σ.
- Valid Shift: A shift s is valid if $T[s+1 \ldots s+m] = P[1 \ldots m]$ where $0 \leq s \leq n-m$.
- Naive Algorithm: Checks all $n - m + 1$ potential shift positions sequentially.
- Naive Worst-Case Time: $O((n-m+1)m)$ when pattern and text contain repeated characters.
- Naive Best-Case Time: $O(n)$ when first character mismatch occurs immediately.

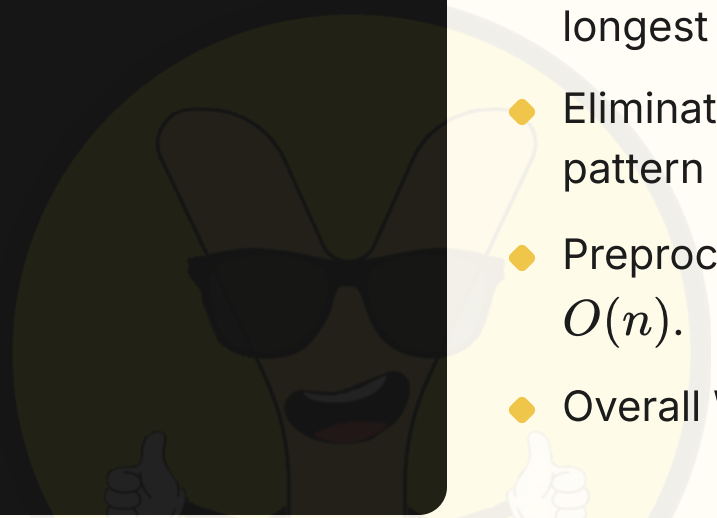
Rabin-Karp String Matching Algorithm

Rabin-Karp leverages rolling hash functions to match patterns in linear average time.

- Hash Function: Computes numerical hash value for pattern and every text window of length m.
- Rolling Hash Property: Calculates hash t_{s+1} from previous hash t_s in $O(1)$ constant time.
- Hash Recurrence: $t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \pmod{q}$.
- Spurious Hits: Occur when hash values match but character sequences differ; verified by character checks.
- Time Complexity: Average time $O(n+m)$, worst-case time $O(nm)$.

Knuth-Morris-Pratt (KMP) Prefix Function

```
def compute_prefix_function(P):
    m = len(P)
    pi = [0] * m
    k = 0
    for q in range(1, m):
        while k > 0 and P[k] ≠ P[q]:
            k = pi[k - 1]
        if P[k] == P[q]:
            k += 1
        pi[q] = k
    return pi
```

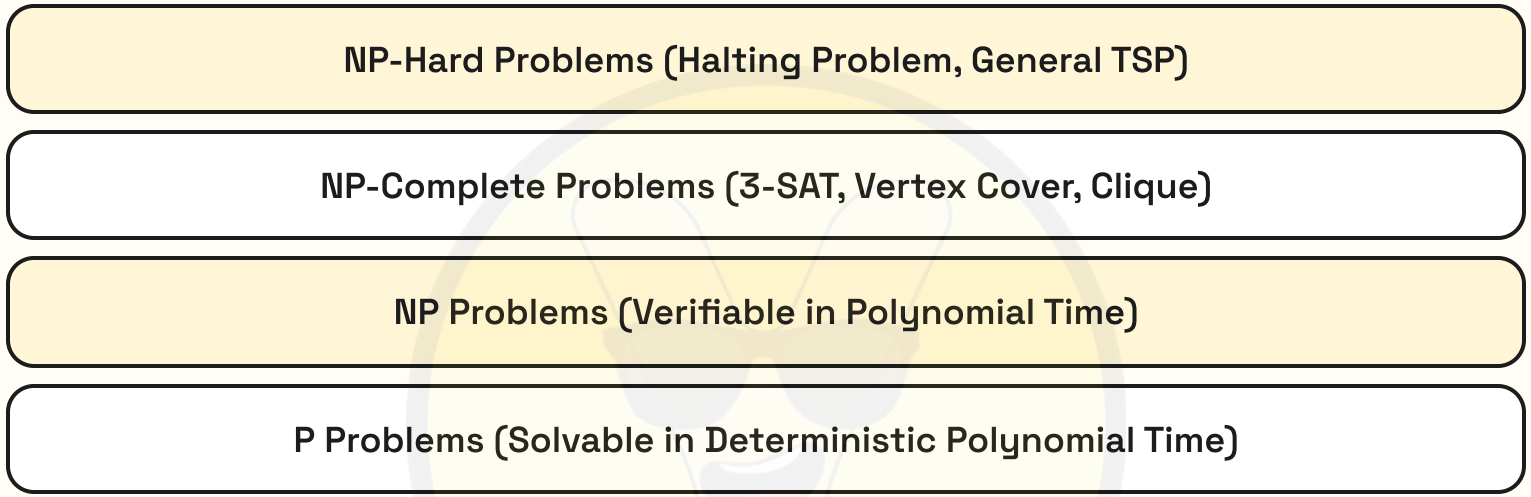


- Computes prefix array π storing length of longest proper prefix that is also a suffix.
- Eliminates redundant comparisons by shifting pattern based on π values.
- Preprocessing Time: $O(m)$; Matching Time: $O(n)$.
- Overall Worst-Case Complexity: $O(n+m)$.

Comparison of String Matching Algorithms

Algorithm	Preprocessing Time	Matching Time	Space Complexity
Naive Algorithm	$O(0)$	$O((n-m+1)m)$	$O(1)$
Rabin-Karp	$O(m)$	$O(n+m)$ avg / $O(nm)$ worst	$O(1)$
Finite Automaton	$O(m\|\Sigma\|)$	$O(n)$	$O(m\|\Sigma\|)$
Knuth-Morris-Pratt (KMP)	$O(m)$	$O(n)$	$O(m)$
Boyer-Moore	$O(m+\|\Sigma\|)$	$O(n/m)$ best / $O(nm)$ worst	$O(\|\Sigma\|)$

Hierarchy of Complexity Classes



Classification of decision problems under the assumption that P != NP.

Theory of NP-Completeness & Reductions

NP-completeness formalizes the computational hardness of decision problems.

- Class P: Decision problems solvable in deterministic polynomial time $O(n^k)$.
- Class NP: Decision problems verifiable in polynomial time given a certificate.
- Polynomial-Time Reduction ($L_1 \leq_P L_2$): Maps instances of L_1 to L_2 in polynomial time.
- NP-Hard: Problem L is NP-hard if every problem in NP polynomial-time reduces to L.
- NP-Complete: Problem L is NP-complete if $L \in \mathrm{NP}$ and L is NP-hard.

Cook’s Theorem & Proving NP-Completeness

Cook-Levin Theorem provides the baseline foundation for proving NP-completeness.

- Cook's Theorem (1971): The Boolean Satisfiability Problem (SAT) is NP-complete.
- Proving Problem X is NP-Complete: Step 1: Prove $X \in \mathrm{NP}$.
- Step 2: Select a known NP-complete problem Y (e.g., 3-SAT, Vertex Cover).
- Step 3: Construct a polynomial-time reduction algorithm $Y \leq_P X$.
- Step 4: Prove correctness: Instance of Y is YES if and only if transformed instance of X is YES.

Approximation Algorithms & Ratio

APPROXIMATION RATIO

An algorithm for an NP-hard optimization problem has an approximation ratio $\rho(n)$ if for any input size n, the cost C of the returned solution satisfies $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$, where C^* is the optimal cost.

- Used to obtain provably good polynomial-time solutions for NP-hard optimization problems.
- Polynomial-Time Approximation Scheme (PTAS): Achieves $(1+\epsilon)$ ratio for any fixed $\epsilon > 0$.
- Fully PTAS (FPTAS): Execution time is polynomial in both input size n and $1/\epsilon$.

2-Approximation Algorithm for Vertex Cover

Find a small vertex cover in undirected graph G=(V,E) using a 2-approximation greedy edge selection strategy.

1. Initialize vertex cover set $C = \emptyset$ and working edge set $E' = E$.
2. While $E' \neq \emptyset$: Choose an arbitrary edge $(u, v) \in E'$.
3. Add both endpoints u and v to vertex cover set C.
4. Remove all edges from E' that are incident to either u or v.
5. Return set C.

Selected edges form a maximal matching M. Optimal cover requires at least |M| vertices, while algorithm picks 2|M| vertices: $|C| \leq 2|C^*|$ (2-approximation).

TSP Approximation via Minimum Spanning Tree

For TSP satisfying Triangle Inequality, an MST yields a polynomial-time 2-approximation.

- Triangle Inequality: Cost function obeys $c(u, w) \leq c(u, v) + c(v, w)$ for all vertices u, v, w.
- Step 1: Compute Minimum Spanning Tree (MST) T of graph G.
- Step 2: Construct full walk W traversing MST edges twice (Eulerian tour).
- Step 3: Form Hamiltonian cycle H by taking shortcuts to bypass already visited vertices.
- Proof Bound: $c(T) \leq c(H^*)$ and $c(W) = 2c(T) \leq 2c(H^*)$; shortcuts only decrease cost, so $c(H) \leq 2c(H^*)$.

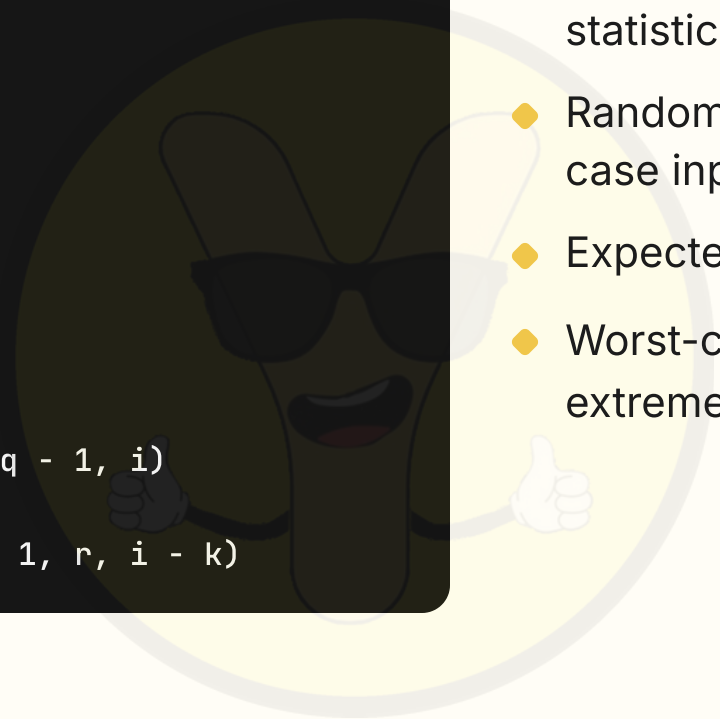
Las Vegas vs. Monte Carlo Randomized Algorithms

Property	Las Vegas Algorithms	Monte Carlo Algorithms
Correctness	Always produces guaranteed correct result	May produce incorrect result with bounded probability
Execution Time	Random variable (worst-case may be unbounded)	Deterministic / strictly bounded upper-bound
Primary Goal	Minimize expected execution time	Minimize failure probability
Error Reduction	N/A (Output is always 100% accurate)	Repeat independent runs to exponentially reduce error
Classic Examples	Randomized QuickSort, QuickSelect	Karger's Min-Cut, Miller-Rabin Primality Test

Randomized QuickSelect Algorithm

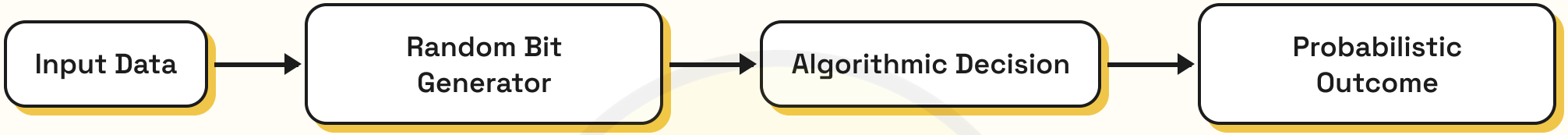
```
import random

def randomized_select(A, p, r, i):
    if p == r:
        return A[p]
    q = randomized_partition(A, p, r)
    k = q - p + 1
    if i == k:
        return A[q]
    elif i < k:
        return randomized_select(A, p, q - 1, i)
    else:
        return randomized_select(A, q + 1, r, i - k)
```



- Finds the i-th smallest element (order statistic) in an array.
- Random pivot selection guards against worst-case inputs.
- Expected Running Time: $O(n)$.
- Worst-case Time Complexity: $O(n^2)$ with extremely low probability.

Randomized Algorithm Execution Flow



Randomized algorithms incorporate pseudo-random choices during execution to break worst-case input biases.

Exam Corner: Unit 5 Important Questions

- **Q1** What is Fast Fourier Transform (FFT)? Explain how FFT reduces polynomial multiplication complexity to O(n log n).
- **Q2** Write and explain the Knuth-Morris-Pratt (KMP) string matching algorithm along with prefix function computation.
- **Q3** Compare Naive, Rabin-Karp, and KMP string matching algorithms in terms of preprocessing time, matching time, and space complexity.
- **Q4** Define complexity classes P, NP, NP-Hard, and NP-Complete. Explain the procedure to prove that a problem is NP-Complete.
- **Q5** State Cook's Theorem and describe how polynomial-time reductions are used to prove 3-SAT is NP-Complete.
- **Q6** Explain the 2-approximation algorithm for the Vertex Cover problem with a worked example and prove its ratio bound.
- **Q7** Differentiate between Las Vegas and Monte Carlo randomized algorithms with suitable examples.

Unit 5 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 1 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

