CS 435

Design and Analysis of Algorithms*:*

"Discovering the Hidden Dynamics of the Laws of Nature"

Dr. Emdad Khan
ekhan@miu.edu
Dept. of Computer Science
Maharishi International University
August 2023

Course: Design and Analysis of Algorithms (*CS 435)*

Vision

- A **Core** Course for CS Program
- Connects Parts to the Wholeness of the CS Program: FPP, MPP, SWE, WAA, Big Data, ML, NLP,..
  - Direct use of Learning from SCI
- Strong Connection to the Industry: Design, Test, Manage S/W,…
- Essential for Growth: Solving Future Problems
- Helping the Society and Leading to Prosperity
- Knowledge, Action, Achievement and Fulfillment

Course Overview Chart

Course Overview Chart CS 435 - Algorithms: Discovering the Hidden Dynamics of the Laws of Nature

Week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1 Theme: How to analyze algorithms	**AM: Lesson 1**: *Introduction: Solving Problems with Algorithms* **PM: Lab 1**	**AM: Lesson 2:** *Intro to Analysis Of Algorithms.* **PM: Lab 2**	**AM: Lesson 2** (continued) **PM: Lab 2**	**AM: Lesson 3**: *Average Case Analysis* **Lab 3** Labs 1, 2 are due Review Solutions	**AM: Lesson 4**: *More Average Case Analysis* **PM: Lab 4**	**AM: Lesson 5:** ***MergeSort***
2 Theme: Sorting Algorithms	**AM: Lesson 6:** *QuickSort* **PM: Lab 6** Labs 3-4 are due Review Solutions	**AM: Lesson 7**: *Lower Bound on Comparison-Based Algorithms And RadixSort* **PM: Lab 7** *End of course material for Midterm Exam*	**AM: Lesson 8**: *Data Structures* **PM: Lab 8** Labs 6, 7 are due, Review Solutions	***AM:*** Review for Midterm **PM:** Study for midterm	***Midterm Exam***	**AM: Lesson 9**: *Binary Search Trees and Enhancements*
3 Theme: Data Structure and Graphs	**AM: Lesson 10**: *Red-Black Trees* **PM: Lab 10** Review MidTerm Solutions	**AM: Lesson 11, 11A**: *Heaps and Priority Queue, Dynamic Prog.* **PM: Lab 11** Labs 8, 10 are due Review Solutions	**AM: Lesson 12: Dynamic prog. (contd.) /** *Introduction to Graph Theory.* **PM: Lab 12**	**AM: Lesson 13** *Implementing Graph Algorithms* **PM: Lab 13**	**AM: Lesson 13 (contd)** *Weighted Graphs – Shortest Path Algorithms* **PM: Lab 13 (contd)** Labs 10 – 12 are due Review Solutions	**AM: Lesson 14:** *Weighted Graphs – Minimum Spanning Tree Algorithms / Advanced Topics (Big Data, ML, Parallel Programming)*
4 Theme: Hard Problems	**AM: Lesson 15** *Hard Problems* **PM: Lab 15**	**AM: Lesson 15** *Hard Prob. (contd)* **Labs 13, 15 are due** Review solutions and review for Final Exam	**AM: Review for Final exam** **PM: Study for final**	**Final Exam**		

Course Syllabus

Goals and Objectives of the Course

- ➢You will Develop skills in designing algorithms and learn to represent algorithms in an implementation-neutral algorithm language
- ➢Learn tools for evaluating efficiency of an algorithm, both empirically and analytically
- ➢Develop the ability to demonstrate the correctness of an algorithm
- ➢Develop discriminative ability in making the optimal selection of a data structure and/or an algorithm in a particular setting
- ➢Learn a core of classical algorithmic solutions to both practical and theoretical problems.
- ➢Become acquainted with techniques for improving efficiency of an algorithmic solution.
- ➢Become acquainted with the self-referral transcendental foundation of all computation both on a theoretical and experiential level

Course Syllabus (short version)

Skills to Develop

1. Analysis of efficiency of an algorithm using
   - a. big-oh, big Omega etc
   - b. worst-case complexity
   - c. average case complexity

2. Techniques of algorithm design
   - a. creating your own algorithms and express in pseudo-code
   - b. using recursion (self reference)
   - c. using design strategies such as "divide and conquer"
   - d. reducing complexity of an existing algorithm

   [Details skipped – see the Syllabus I emailed]

Course Description

- This course presents methods for analyzing the efficiency of algorithms (including worst-case and average-case analysis) and introduces a variety of known, highly efficient algorithms. Analysis, design, and implementation of algorithms are given equal emphasis. Topics include searching and sorting, efficiency of operations on data structures (including lists, hashtables, balanced binary search trees, priority queues), graph algorithms, combinatorial algorithms, recurrence relations, Dynamic Programming, NP-complete problems, and some special topics as time allows
- (Special topics include computational geometry, algorithms for cryptosystems, approximation, Big Data and parallel computing). (4 units) *Prerequisites:* CS 401 and MATH 272, or consent of the department faculty.

Student Learning Chart

ALGORITHMS (CS 435) STUDENT LEARNING CHART

Course Objectives, ACTIVITIES, AND ASSESSMENTS

OBJECTIVES **This is what** **you'll learn to do***	LEARNING ACTIVITIES This is how you'll learn it	ASSESSMENTS This is how you'll show you've learned
1. Develop skills in designing algorithms and learn to represent algorithms in an implementation-neutral algorithm language **(1, 4, 7)**	By working individually, in teams and practicing the techniques to understand, analyze and design key algorithms using pseudocode	Lab works and results from the midterm exam
1. Learn tools for evaluating / improve efficiency of an algorithm, both empirically and analytically **(6, 7, 8)**	By practicing the techniques using counting primitive operations, big-Oh, big Omega, Big Theta and b. worst-case complexity c. average case complexity in homework as well as in labs	Lab works and results from the midterm exam
3. Develop the ability to demonstrate the correctness of an algorithm **(4, 5, 7)**	By practicing the Proof of Correctness techniques in classroom, labs and homework	Lab works and results from the midterm exam
4. Develop discriminative ability in making the optimal selection of a data structure and/or an algorithm in a particular setting **(4, 5, 7)**	By practicing the techniques of using & analyzing of various Data Structures in various algorithms in classroom, labs and homework	Lab works and results from the midterm & final exams
5. Learn a core of classical algorithmic solutions to both practical and theoretical problems. **(5, 9)**	By practicing the techniques like Sorting, Searching, Dynamic Programming and Graph Algorithms in classroom, labs and homework	Lab works and results from the midterm & final exams
6. Learn the definition and analysis of Hard problems **(1, 3, 4)**	By practicing the techniques in classroom, labs and homework	Lab works and results from the final exam
7. Learn the connections between the Science of Consciousness and Algorithm **(1, 3, 4, 9)**	By doing Wholeness of the Lessons, Main Points, Unity Charts and associated explanations in classroom	Results from the final exam

Teaching, Learning and Interaction

1. Very Interactive - to ensure effective teaching and learning
 [Related to Maharishi's Principles of Ideal Learning - Reception, Intelligence, Knowledge, Experience (examples, labs), Expressions – how you express what you learned]

2. Highly encourage to ask questions. Unless it is a burning question, wait until a critical part of a topic is explained

3. Think of new (similar) problems for algorithm / topic covered (a good way to learn)

4. Group discussion, especially in the lab

5. 3 types of questions, problems etc:
   - easy - medium – a bit challenging
   (to challenge your thinking, to help learn more….)

6. Will Use Industry Examples Frequently to ensure better Learning

Class Schedule and Grading

- **Text Book**

1. The strongly recommended text for the course is *Algorithm Design: Foundations, Analysis, and Internet Examples,* by Michael Goodrich and Roberto Tamassia
2. *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein - Third edition

- **Grading**

1. Mid Term - 46%
2. Final - 46%
3. Lab (Group work), attendance, class participation – 8%

- **Consciousness-Based Education**
  - Develop Full Potential, Integrate Subjective & Objective
  - Proper Use of TM
- **Academic Honesty is Very Important**

Lesson 1: Introduction: Solving Problems with Algorithms

Wholeness of the Lesson

Algorithm, a procedure or sequence of steps for any computation, is the hidden building block of all computing systems; study of algorithm enables one to develop applications using software, hardware or their combinations.

Science of Consciousness: *Transcendental Meditation (TM) allows one to easily reach* the state of pure creative intelligence from where the un-manifest abstract ideas and thoughts may be efficiently converted into the fully expressed useful values and objects.

Algorithms As Concentrated Intelligence

- An algorithm is a procedure or sequence of steps for computing outputs from given inputs i.e. solving a problem
- Algorithms can be implemented in programming languages like Java, C, Python and the like
- We will study techniques for creating algorithms, measuring their efficiency, and refining their performance
- In this lesson, we show that, as remarkable as the many feasible algorithms that have been found are, (and as deep as the intelligence is behind such solutions), the truth is that the vast majority of "problems" that could be solved remains – will always remain – completely beyond the reach of any kind of solution.

Examples of Problems

1. (***GCD Problem***) Given two positive integers m, n, is there a positive integer d that is a factor of both m and n and that is bigger than or equal to every integer d’ that is also a factor of m and n?
2. (***Sorting Problem***) Given a list of integers, is there a rearrangement of these integers in which they occur in ascending order?
3. (**Autonomous Driving Problem**) How a car can be driven by a machine? [**Machine Learning Algorithm is Needed**]
4. (***Subset Sum Problem***) Given a set S of positive integers and a nonnegative integer k, is there a subset T of S so that the sum of the integers in T equals k?
5. (*n x n Chess Problem*) Given an arbitrary position of a generalized chess-game on an n × n chessboard, can White (Black) win from that position?

1. (***Halting Problem***) Given a Java program R (having a main method), when R is executed, does R terminate normally? (No runtime exceptions, no infinite loops.)

Some History

In 1928 in the world of mathematics

- There was hope that a set of axioms (rules) could be identified that could unlock all the truths of mathematics
- Properly applied, these rules could be applied to solve any math problem

- …and the world would be a better place.

Key Figures

- David Hilbert, Kurt Godel, Alan Turing
- Hilbert Tried to find a general algorithmic procedure (a set of rules) for answering all mathematical inquiries

Hilbert’s Agenda

Three questions at the heart of his agenda were:

- Is mathematics *consistent?*
  - Can no statement ever be proven both true and false with the rules of math?
- Is mathematics *complete?*
  - Can every assertion either be proven or disproven with the rules of math?
- Is mathematics *decidable?*
  - Are there definite steps that would prove or disprove any assertion?

Incompleteness Theorem

- Kurt Gödel wrote a paper in 1931 that shook the math world.
- Proved that consistency and completeness in math could not be attained.
- There is no consistent and complete system of formal rules that is comprehensive enough to include arithmetic.
- SCI point: Total Knowledge cannot be fully described in finite relative terms.
- Another important math question:

  Is mathematics *sound?*

  - Is every theorem (proven statement) also "True"? E.g. P ^ not (p) can be proved but the result is "False". Hence, it is not sound.

    (Converse of Completeness – Can we prove Every Statements that are "true"?)

Arithmetic is Incomplete

- Can we come up with enough number of Axioms that can prove all True properties of Model N, the non-negative Integers?
- Consider the following Axioms of NT
  - $\forall x\ (f(x) != 0)$
  - $\forall x\ (x + 0 = x)$
  - $\forall x\ (x.o = 0)$
  - $\forall x\ (x < f(x)$ where f is Unary function i.e. $f(x) = x + 1$.
  - …………..
- Unfortunately a complete list of axioms like above does NOT exist to make Number System / Arithmetic Complete.

Decidability Problem

- Alan Turing, sometimes called the “father of computer science”.
- Studied Godel’s work in 1935
- Studied the Problem:

Can one find an algorithm to determine whether a mathematical proposition is true or false or are some propositions undecidable?

Halting Problem

- *Proposition: Given a description of a Turing machine and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting.*
- Recursive – one Turing machine analyzes another Turing machine
- More on this – a few slides later

[These are all part of "Complexity Theory" class – so will not go into very details]

Known Algorithms – P Problems

1. (GCD Problem) The Euclidean Algorithm computes the gcd of two positive integers m <= n. It requires fewer than n steps to output a result. We say it *runs in O(n) time*.
2. (SORTING Problem) MergeSort is an algorithm that accepts as input an array of n integers and outputs an array consisting of the same array elements, but in sorted order. It requires fewer than n2 steps to output a result. We say it runs in O(n2) time.

Both Problems 1 and 2 have algorithms that run in *polynomial time;* this means that the number of steps of processing required is a polynomial function of the input size.

If a problem can be solved using an algorithm that runs in polynomial time, the problem is said to be a *P Problem.*

P Problems are said to have *feasible solutions* because an algorithm that runs in polynomial time can output its results in an acceptable length of time, typically.

Known Algorithms – NP and EXP Problems

3. (SUBSETSUM Problem) Every known algorithm to solve SubsetSum (with input array containing n elements) requires approximately 2^n steps of processing ("exponential time") to output a result (in the worst case). However, it is still possible that someone will one day find an algorithm that solves SubsetSum in polynomial time.
4. (N X N CHESS Problem) Like SubsetSum, every known algorithmic solution requires exponential time in the worst case. However, it has also been shown that no algorithmic solution could ever be found that runs in polynomial time.

- Both SubsetSum and n x n Chess are problems that belong to **EXP**. This means that each has an exponential time algorithm that solves it. The possibility remains that SubsetSum may also belong to the smaller class P, but this question remains unsolved. Read about the class **EXP** at http://en.wikipedia.org/wiki/EXPTIME
- n x n Chess is a special type of problem in **EXP**, known as **EXP-complete**. For this lecture, it suffices to say that **EXP** -complete means that there is no way to devise an algorithm that solves it in polynomial time. See

  http://www.ms.mff.cuni.cz/~truno7am/slozitostHer/chessExptime.pdf

When the only known solutions to a problem are exponential, the problem is said to *have* ***no known feasible solution.***

Known Algorithms – NP and EXP Problems

- The SubsetSum Problem is also an NP Problem: This means that there is an algorithm A and an integer k so that, given a solution T to a SubsetSum problem instance with input size n, A can verify that T is indeed a correct solution and does so in fewer than n^k steps (i.e. in polynomial time)
- Sample Verification:

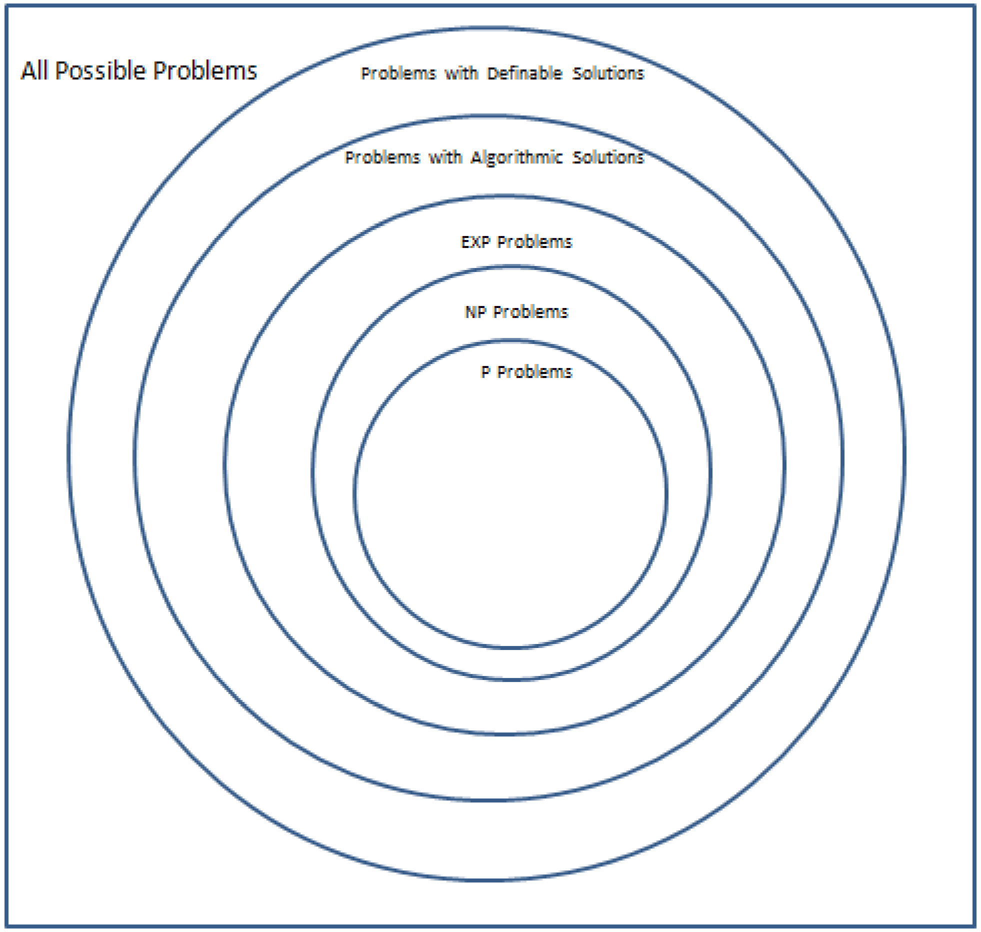
  Start with a SubsetSum problem instance S = $\{s_1, s_2, . . ., s_n\}$ and k, where $s_1, \ldots, s_n$ and k are all positive integers. Given also a solution T: A solution is a subset of S whose elements add up to k.

  Verification:

  ```
  sum = 0;
  for each j in T
      sum += j
  if(sum == k) return true;
  else, return false;
  ```

- n x n Chess is not known to belong to NP (but one day someone may prove that it does belong to NP). In general, every NP problem belongs to EXP, but whether NP = EXP is not known.   See http://en.wikipedia.org/wiki/EXPTIME

Classes of Problems


All Possible Problems
Problems with Definable Solutions
Problems with Algorithmic Solutions
EXP Problems
NP Problems
P Problems

The HALTING Problem (already shown)

The Problem: Given a Java program R (having a main method), when R is executed, does R terminate normally?

It is known that *there is no algorithm that could possibly solve the Halting Problem.* Although it is possible to *define* a solution, there is no procedure that could actually *compute* a solution.

Unsolvability of the Halting Problem

Begin with an observation: Every Java program may be encoded as a positive integer in such a way that, from the integer code, the program can be reconstituted.

Unsolvability 2

Encoding Java programs

- Assume < 512 distinct characters are ever used in a Java program, so each character can be represented by a bit string of length 9
- Encode a program by eliminating white space, start with ‘1’, and concatenate the encoded characters one-by-one

```
BigInteger f(BigInteger[] n){
    return n[0].add(n[0]);
}
```

Char	Code	Char	Code	Char	Code
a	000000001	d	000000010	e	000000011
f	000000100	g	000000101	i	000000110
n	000000111	r	000001000	t	000001001
u	000001010	B	000001011	I	000001100
.	000001101	{	000001110	}	000001111
(	000010000	)	000010001	;	000010010
⟨space⟩	000010011	[	000010100	]	000010101
0	000010110				

```
1000001011000000110000000101000001100000000111000001001000000001100000101
 000000011000001000000001001100000010000001000000000101100000001100000000101
 000001100000000011100000100100000000110000001010000000110000010000000010100
 000010101000010011000000111000010001000000111000000100000000000011000001001
 000001010000000100000000011100001001100000011100001010000000101100000010101
 000001101000000000100000001000000000100000100000000000111000010100000010110
 000010101000010001000010010000001111
```

Unsolvability 3

- With this encoding, we can state the Halting Problem in terms of a function H:

  H(e,n) = 1 if the program

  encoded by e halts when run

  on input n; else H(e,n) = 0.
- The Halting Problem is: What are the output values of H for all inputs e, n?
- A solution for the Halting Problem is *definable* since we can specify a function H that solves it. (More precisely: It can be shown that H is definable in the standard model of arithmetic; if we could know all true statements of arithmetic, we would have all information about H)
- The important question is: Is there an algorithm A that computes the values of H (so that, on input e, n, A outpus H(e,n))? The Halting Problem is said to be *unsolvable* because no such algorithm exists.

Unsolvability 4

- Suppose H is computable. Define another function G as follows:

  G(e) = 1 if H(e,e) = 0

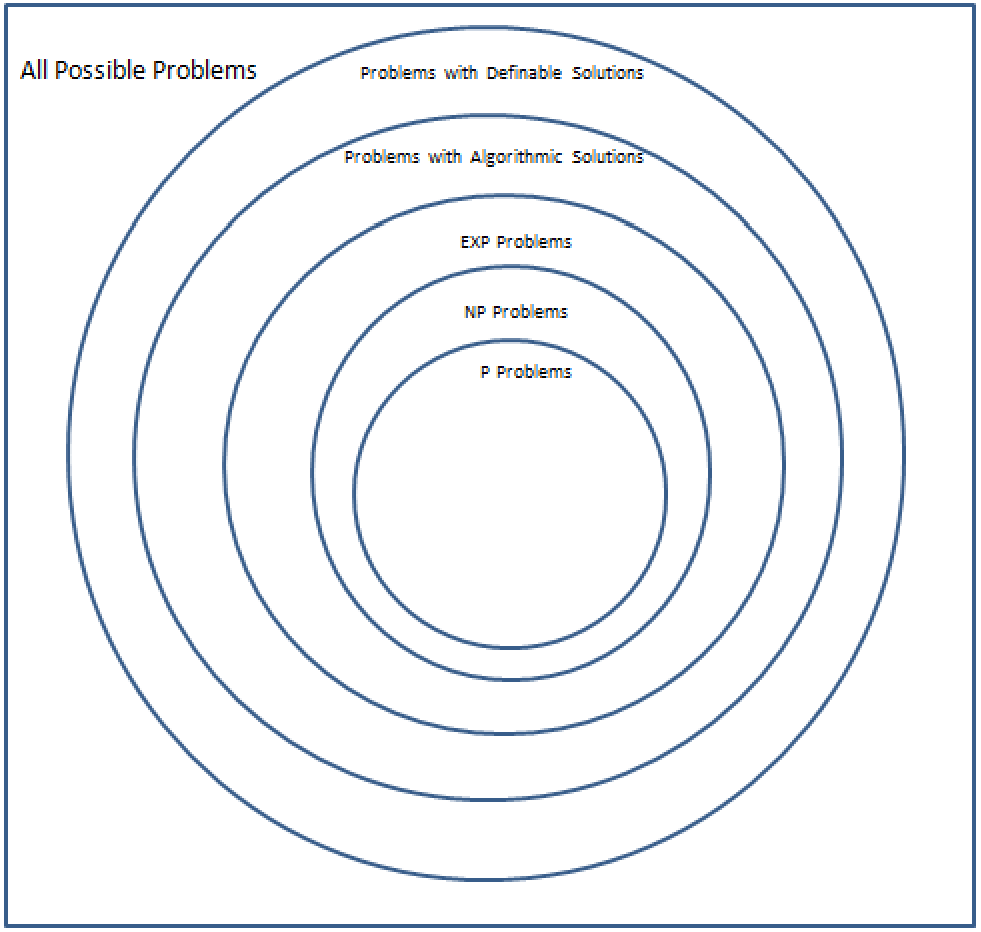
  G(e) is undefined if H(e,e) = 1

- *If H is computable, so is G*: Let P be a program that computes values of H. Create a program Q that computes G like this: On input e, Q runs P on e. If P outputs 0, Q outputs 1 and halts. If P outputs 1, Q goes into an infinite loop. This shows G is computable.

Unsolvability 5

- Use computability of G to obtain a contradiction. Suppose u is the integer that encodes the program Q. We run Q on input u and ask: Does Q halt on input u?
- Suppose Q does eventually halt on input u; then it outputs 1 on this input. By definition H(u,u) = 0. But this means that Q when run on input u, does *not* halt. Impossible.
- Suppose Q does *not* halt on input u. Then H(u,u) =1, which means that when Q is run on input u, it *does* halt. Impossible
- This shows that the assumption that H is computable leads to absurd conclusions. Therefore, H is *not* computable.

Classes of Problems


All Possible Problems
Problems with Definable Solutions
Problems with Algorithmic Solutions
EXP Problems
NP Problems
P Problems

The Full Range of “Problems”

- The Halting Problem has a *definable* solution because there is a definable function H (definable in the standard model of arithmetic) that answers all questions about whether a given Java program terminates on a given input
- Likewise, the abstract notion of “problem” is related to the concept of a function. Speaking generally, every definable function corresponds to (and is a definable solution for) a “problem”.
- More generally, *every* function from f: N -> N (whether or not definable) specifies a problem. But *most* such functions are *not* definable – therefore, most “problems” cannot even be formulated precisely in the language of mathematics.

Summary

- Algorithm Connects all Course of CS
- Very Key for your Success in the Industry
- You will learn how to design and analyze algorithms
- SCI Can be the Key to Get Full Development of Knowledge and Intelligence In Algorithms and CS.
- Most Algorithms Used in Industry are P-Problems
- Many More Very Complex Algorithms Exist (e.g. EXP) many of which Cannot be Solved Today. Using the Knowledge of SCI and Cosmic Computing We would be able to solve such problems in Future

Main Point

It has turned out that the problems that are most needed to be solved for the purpose of developing modern-day software projects happen to lie in the very specialized class of P Problems – problems that have feasible solutions. For these problems, the creativity and intelligence of the industry has been concentrated to a point; the algorithms that have been developed for these are extremely fast and highly optimized. At the same time, this tiny point value reveals how vast is the range of problems that cannot be solved in a feasible way. **These points illustrate that creative expression arises in the collapse of unboundedness to a point, and also that unboundedness itself is beyond the grasp of the intellect.**

Unity Chart: Connecting the Parts of Knowledge With The Wholeness of Knowledge

Value of Algorithms in Computers Science

1. The algorithms that are used in practice in the development and analysis of software runs in *polynomial time.*

2. Although polynomial algorithms are the main algorithms used in practice, there exist many more complex problems, and complex algorithms would be needed to solve such problems.

Unity Chart (Contd)

- *Transcendental Consciousness* is the fully expanded field of consciousness, beyond even the most general notion of "algorithm".
- *Impulses Within The Transcendental Field*. As pure consciousness becomes conscious of itself, it undergoes transformational dynamics, as knower, known, and process of knowing interact. The process of knowing is the first sprout of the notion of "algorithm"
- *Wholeness Moving Within Itself.* In Unity Consciousness, the transformational dynamics of consciousness, at the basis of all of creation, are appreciated as the lively impulses of one's own consciousness, one's own being.