Lesson 1: *Introduction to Algorithms*

Creation Emerging from the Collapse of Wholeness to a Point

Wholeness of the Lesson

Much of the power of modern software arises from the fact that it makes use of highly efficient algorithmic solutions to a variety of problems. These solutions represent the concentrated intelligence of great thinkers of the past half-century. Yet the range of problems for which such solutions are known is actually only a tiny speck in the landscape of abstract problems. Likewise, the full range of Nature's computational power is too vast to be grasped by the intellect alone, yet harnessing even a little of that power is enough to produce powerful results.

Algorithms As Concentrated Intelligence

- An algorithm is a procedure or sequence of steps for computing outputs from given inputs
- Algorithms can be implemented in a programming language such as Java, C++, Scala, Python and so on.
- 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 that is behind such solutions, the truth is that the vast majority of "problems" that could be solved remains – and will always remain – completely beyond the reach of any kind of solution that could in principle be implemented on a computer.

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?
   https://www.youtube.com/watch?v=JUzYl1TYMcU

https://randerson112358.medium.com/euclides-algorithm-gcd-fd664f5ac540

```
static int gcd(int a, int b)
{
        if(b == 0)
        {
                return a;
        }
        return gcd(b, a % b);
}
```

2. (*Sorting Problem*) Given a list of integers, is there a rearrangement of these integers in which they occur in ascending order?

Examples of Problems

3. (*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?

   int[] S = { 3, 2, 7, 1}, k = 6
   Output: True, subset is {3, 2, 1}

4. (*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?

5. (*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.)

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 n^2 steps to output a result. We say it runs in $O(n^2)$ 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***. (A problem that can be solved in polynomial time using a deterministic Turing machine.)

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(A problem that can be solved in polynomial time using a nondeterministic Turing machine.): This means that there is an algorithm A and an integer t 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^t steps (i.e. in polynomial time). (That is., Polynomial verifiability)
- Sample Verification:

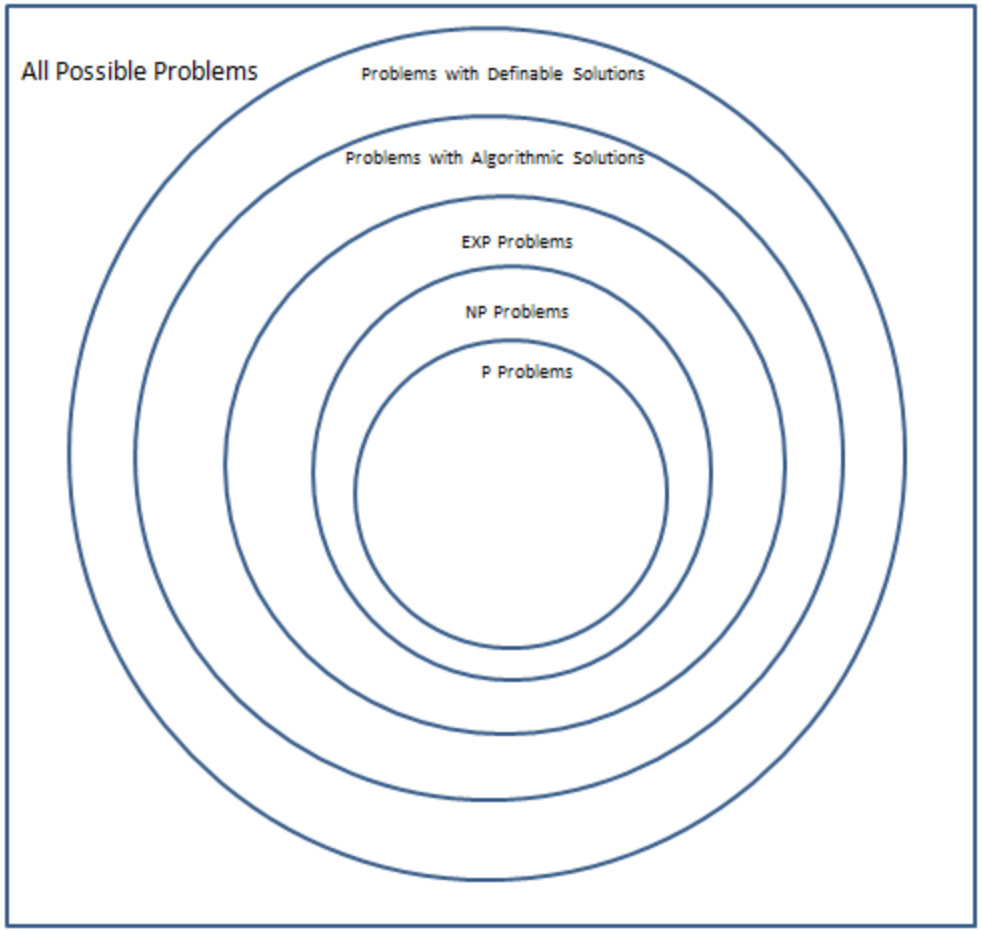
  Start with a SubsetSum problem instance S = $\{s_1, s_2, \ldots, s_n\}$ and k, where $s_1, \ldots, s_n$ and k are all positive integers. Given also a solution T: A solution is a subset of S whose elements add up to k.

  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



The HALTING Problem

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.

https://www.youtube.com/watch?v=92WHN-pAFCs

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				

1000001011000000110000000101000001100000000111000001001000000011000000101
000000011000001000000010011000000100000010000000001011000000110000000101
000001100000000111000001001000000011000000101000000011000001000000010100
000010101000010011000000111000010001000001110000001000000000011000001001
000001010000001000000000111000010011000000111000010100000010110000010101
000001101000000001000000010000000010000010000000000111000010100000010110
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

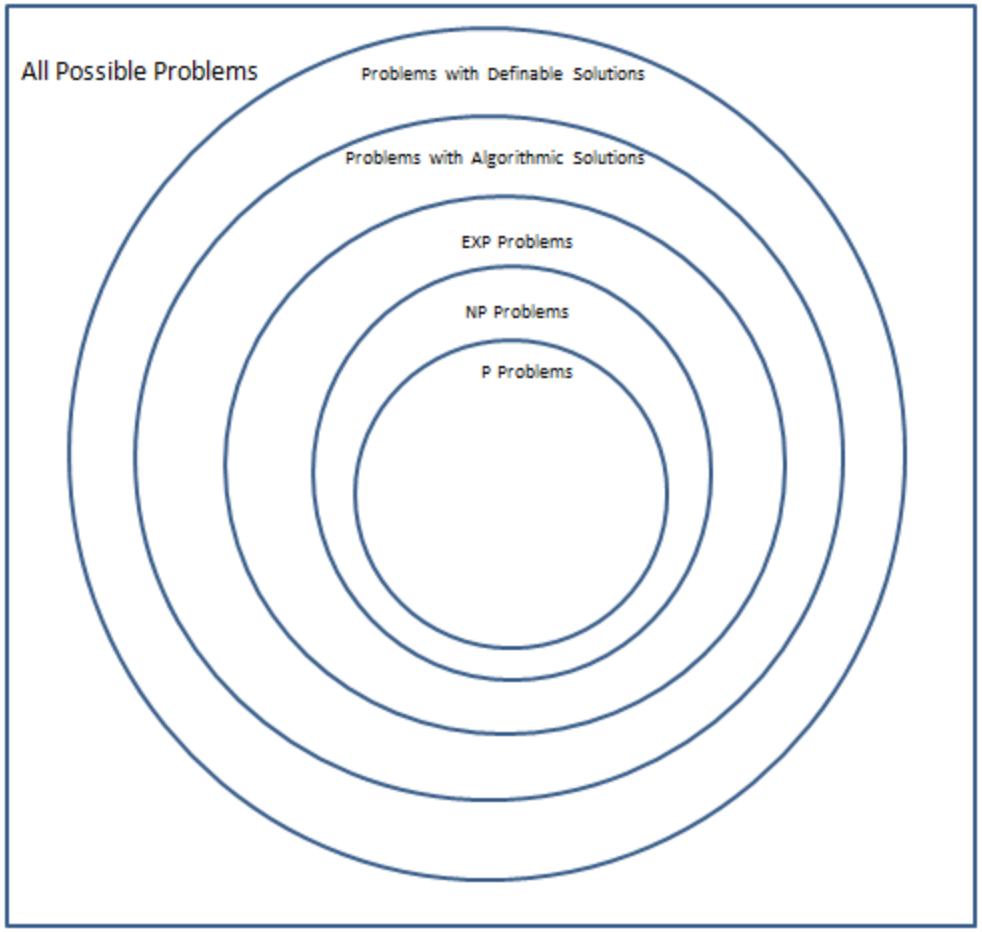
  G(e) is undefined if H(e,e) = 1

- *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



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.

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.

Connecting the Parts of Knowledge With The Wholeness of Knowledge

1. The functions that are used and analyzed in practice in the development and analysis of software are the *polynomial bounded functions.*
2. The polynomial bounded functions form only a tiny speck in the expansive class of all number-theoretic functions N → N.

3. *Transcendental Consciousness* is the fully expanded field of consciousness, beyond even the most general notion of "function".
4. *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 "function."
5. *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.