Introduction to Computation

- What is Computation
- Paper-Pencil Arithmetic and Compass-Ruler Geometry
- Why Study Theory of Computation – The Computational Lens
- Formalizing Computation
- Computability
- Complexity
- Cryptography

Imdad ullah Khan

What is Computation?

compute verb

com·pute kəm-ˈpyüt

computed; computing

transitive verb

: to determine especially by mathematical means

Compute the area of the triangle.

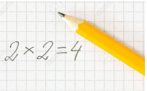
Merriam-Webster

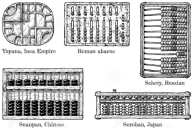
Notice no mention of a device

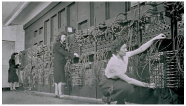
> Computer science is no more about computers than astronomy is about telescopes
>
> Edsger Dijkstra

What is Computation?

Computation: Processing information by applying a finite set of rules







Paper + Pencil Arithmetic

$$
\begin{array}{cccccccc}
 & & 1 & & 1 & 1 & & \\
 & 4 & 6 & 9 & 2 & 7 & 5 & 8 \\
+ & 5 & 1 & 7 & 2 & 2 & 6 & 1 \\
\hline
 & 9 & 8 & 6 & 5 & 0 & 1 & 9
\end{array}
$$

$$
\begin{array}{cccccc}
 & & & 7 & 5 & 8 \\
 & & \times & 6 & 3 & 2 \\
\cline{3-6}
 & & 1 & 5 & 1 & 6 \\
 & 2 & 2 & 7 & 4 & \\
4 & 5 & 4 & 8 & & \\
\hline
4 & 7 & 9 & 0 & 5 & 6
\end{array}
$$

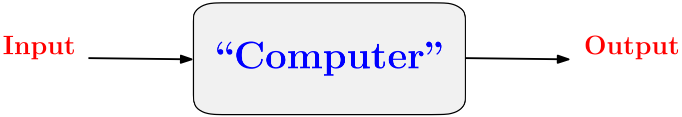
Single Digit Multiplication Lookup Table

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	50	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Rules to process the information implicitly give tools and power available

What is Computation?

Computation: Processing information by applying a finite set of rules



Description of Processing is called Algorithm that converts the input to the desired output

Different set of rules/operations lead to different computational capabilities and limits

Information needs to be encoded to be input for application of rules/operations

Why Study Theory of Computation?

1. To learn new ways of thinking about computing

We learn general ideas that can be applied to many models of computation expressed abstractly and precisely

Abstractly: independent of technology, applies both to present and future

▷ Suppress inessential implementation level details

Precisely: means can formally prove

Positive Results: What is computable, correctness of algorithms/systems

Negative Results: What is not computable/not computable in fixed resources

Why Study Theory of Computation?

1. To learn new ways of thinking about computing

We learn general ideas that can be applied to many models of computation expressed abstractly and precisely

What can (not) be computed? ▷ Computability

What can (not) be computed using a fixed resources? ▷ Complexity

Can we say Problem X is “harder” than Problem Y?

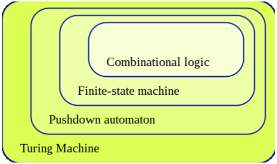
Is there a single computer that can simulate every other computer?

▷ Universal Computer

Why Study Theory of Computation?

1. To learn new ways of thinking about computing
2. To formalizes different models of computational devices

- Finite Automata
- Pushdown Automata
- Stream Computer
- Turing Machines
- Quantum Computer
- Parallel Computer
- Distributed Computers



Karp R M. Understanding science through the computational lens. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(4): 569–577 July 2011. DOI 10.1007/s11390-011-1157-0

Understanding Science Through the Computational Lens

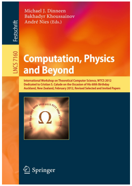
Richard M. Karp

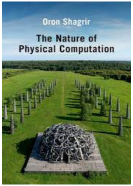
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720-1776, U.S.A.

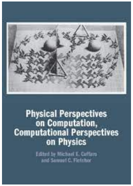
E-mail: karp@cs.berkeley.edu

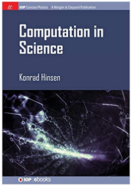
Received December 18, 2008; revised April 29, 2011.

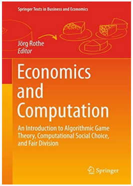
Abstract This article explores the changing nature of the interaction between computer science and the natural and social sciences. After briefly tracing the history of scientific computation, the article presents the concept of *computational lens*, a metaphor for a new relationship that is emerging between the world of computation and the world of the sciences. Our main thesis is that, in many scientific fields, the processes being studied can be viewed as *computational* in nature, in the sense that the processes perform dynamic transformations on information represented as digital data. Viewing natural or engineered systems through the lens of their computational requirements or capabilities provides new insights and ways of thinking. A number of examples are discussed in support of this thesis. The examples are from various fields, including quantum computing, statistical physics, the World Wide Web and the Internet, mathematics, and computational molecular biology.

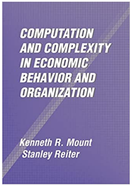












Formalizing Computation

We have been computing for thousands of years

Input: Two n digits arrays A and B
Output: (integer) $C = A \times B$

Algorithm Integer Multiplication

$C \leftarrow 0$
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    $C \leftarrow C + 10^{i+j} A[i] * B[j]$

Input: Two integers a and b
Output: (integer) $C = \text{GCD}(a, b)$

Algorithm GCD Computation

function $\text{GCD}(a, b)$
  if $b = 0$ **then**
    return a
  else
    $r \leftarrow a \,\%\, b$
    return $\text{GCD}(b, r)$

But computation was formalized only recently

Formalizing Computation

Hilbert's 10th problem (1900)

Devise a **finite procedure** to check if a diophantine has integral solution

diophantine equation (e.g. multivariate polynomial) with integer coefficients

e.g. $ax + by = c, \qquad aw^4 + bx^4 + cy^4 + dz^4 = 0 \qquad a, b, c, d \in \mathbb{Z}$

Entscheidungsproblem [Hilbert and Ackermann (1928)]

Devise a **finite procedure** to determine the validity of a logical expression

$$\neg\exists\, x, y, z \in \mathbb{Z} \;:\; (x^n + y^n = z^n) \wedge (n \geq 3)$$

Can Mathematics be mechanized? $\triangleright$ automatic theorem proving

Formalizing Computation

Alonzo Church (1935/1936)

Lambda Calculus is a reasonable notion of **finite procedure** "=algorithm"

AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

By Alonzo Church.

Alan Turing (1936)

Turing Machine is a reasonable notion of **finite procedure** "=algorithm"

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

Alan Turing (1937)

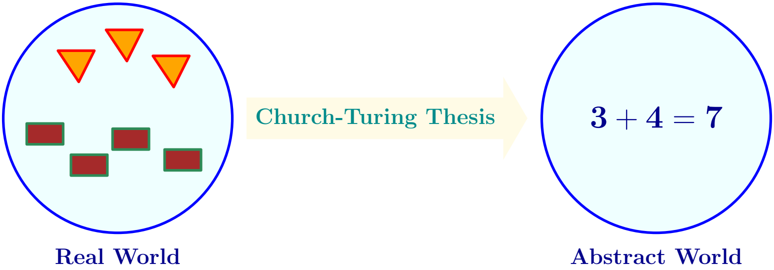
Turing Machine $=$ Lambda Calculus

Formalizing Computation

Church-Turing Thesis

Any computational problem that can be solved by a physical device, can be solved by a Turing Machine

"Computable" $=$ "Computable by a Turing Machine"



Formalizing Computation

Hilbert's 10th problem (1900)

Devise a **finite procedure** to check if a diophantine has integral solution

Matiyasevich-Robinson-Davis-Putnam (1970)

There is no algorithm to solve this problem

Entscheidungsproblem [Hilbert and Ackermann (1928)]

Devise a **finite procedure** to determine the validity of a logical expression

Turing (1936)

There is no algorithm to solve this problem

Two main questions in theoretical computer science

Is there an algorithm to solve a problem ▷ Computability

Just saw two examples of negative answers

Is there an **efficient** algorithm to solve a problem ▷ Complexity

Efficiency is measured by requirements of runtime, memory, number messages passed, random bits, quantum resources, energy, ...

Camp-1: Algorithm Designers

Coming up with efficient algorithms

Camp-2: Complexity Theorists

Proving no efficient algorithm exists

Two main questions in theoretical computer science

Is there an algorithm to solve a problem ▷ Computability

Just saw two examples of negative answers

Is there an **efficient** algorithm to solve a problem ▷ Complexity

Efficiency is measured by requirements of runtime, memory, number messages passed, random bits, quantum resources, energy, ...

Camp-1: Algorithm Designers

Coming up with efficient algorithms

Camp-3: Cryptographer (e.g.)

Camp-2: Complexity Theorists

Proving no efficient algorithm exists

Using Camp-2 results to solve Camp-1 Problems

The Halting Problem

Input: A computer program A.CPP

Output: **Yes** if A.CPP halts on every legal input, else **No**

Compilers and interpreters take programs as input and "analyze" them

This problem has many applications

Computability: Halting Problem

How to check if a program halts

Algorithm Clearly halts

return true

Algorithm Halts if $n \geq 0$ and even

while $n \neq 0$ **do**
  $n \leftarrow n - 2$

Algorithm Never halts

$n \leftarrow 1$
while $n \neq 0$ **do**
  $n \leftarrow n + 1$

Computability: Halting Problem

Algorithm Collatz Program (integer n)

---

while $n \neq 1$ **do**
  if n is even **then**
    $n \leftarrow {}^{n}/_{2}$
  else
    $n \leftarrow 3n + 1$

$n = 3 \implies 3, 10, 5, 16, 8, 4, 2, 1$

$n = 4 \implies 4, 2, 1$

$n = 5 \implies 5, 16, 8, 4, 2, 1$

$n = 6 \implies 6, 3, 10, 5, 16, 8, 4, 2, 1$

$n = 7 \implies 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 15, 8, 4, 2, 1$

$n = 8 \implies 8, 4, 2, 1$

$n = 9 \implies 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 15, 8, 4, 2, 1$

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Computability: Halting Problem

Algorithm Collatz Program (integer n)

while $n \neq 1$ **do**
  if n is even **then**
    $n \leftarrow n/2$
  else
    $n \leftarrow 3n + 1$

Collatz Conjecture (1937)

For every integer n this program eventually reaches 1 (thus halts)

aka wondrous numbers, $3n + 1$ conjecture, Syracuse problem, Ulam conjecture

> **For about a month everyone at Yale worked on it, with no result. A similar phenomenon happened when I mentioned it at the University of Chicago. A joke was made that this problem was part of a conspiracy to slow down mathematical research in the U.S.**
>
> **Shizuo Kakutani, 1960**

> **Mathematics is not yet ripe enough for such questions.**
>
> **Paul Erdös, 1983**

Computability: Halting Problem

Fermat's Last Theorem (1637)

For $n \geq 3$, $a^n + b^n = c^n$ has no solution where a, b, c are positive integers

Algorithm NEGFLT(integer n)

```
flag ← true
a ← 1
while flag = true do
    for b = 1 → a do
        for c = 2 → a + b do
            if a^n + b^n = c^n then
                flag ← false
    a ← a + 1
```

$\text{HALT}(\text{NEGFLT}(n)) =$ **Yes** $\iff$ Fermat's last theorem is false

Ok! we know FLT is true, how about some other

Computability: Halting Problem

Goldbach Conjecture (1742)

Every even integer $n > 2$ is the sum of two primes.

Algorithm $\textsc{negGoldbach}($ even integer $n)$

```
flag ← true
n ← 2
while flag = true do
    flag ← false
    n ← n + 2
    for p = 2 → n do
        if isPrime(p) AND isPrime(n − p) then
            flag ← true
            break
```

$\textsc{halt}(\textsc{negGoldbach}(n)) =$ **Yes** $\iff$ Goldbach conjecture is false

An algorithm for $\textsc{halt}(\cdot)$ would resolve the Goldbach conjecture

Complexity

Multiplying two n digits integers

Algorithm Repeated Addition

$C \leftarrow 1$
for $i = 1$ to B **do**
  $C \leftarrow C + A$
return C

$$A * B = \underbrace{A + A + \ldots + A}_{B \text{ times}}$$

Each addition takes $O(n)$ single digit addition, number of addition is B

Total runtime is $O(n10^n)$ $\triangleright \because B$ could be 10^n

Complexity

Multiplying two n digits integers

Algorithm Long Multiplication

for $i = 1$ to n **do**
  $c \leftarrow 0$
  for $j = 1$ to n **do**
    $Z[i][j+i-1] \leftarrow (A[j] * B[i] + c) \bmod 10$
    $c \leftarrow (A[j] * B[i] + c)/10$
  $Z[i][i+n] \leftarrow c$
$carry \leftarrow 0$
for $i = 1$ to $2n$ **do**
  $sum \leftarrow carry$
  for $j = 1$ to n **do**
    $sum \leftarrow sum + Z[j][i]$
  $C[i] \leftarrow sum \bmod 10$
  $carry \leftarrow sum/10$
$C[2n+1] \leftarrow carry$

$$
\begin{array}{cccccc}
 & & & 7 & 5 & 8 \\
 & & \times & 6 & 3 & 2 \\
\hline
 & & 1 & 5 & 1 & 6 \\
 & 2 & 2 & 7 & 4 & \\
4 & 5 & 4 & 8 & & \\
\hline
4 & 7 & 9 & 0 & 5 & 6
\end{array}
$$

Total single digit arithmetic ops: $O(n^2)$

Complexity

Multiplying two n digits integers

$$AB = (10^n w + x)(10^n y + z) = 10^{2n} \underbrace{(wy)}_{\text{1 multiplication}} + 10^n \underbrace{(wz + xy)}_{\text{2 multiplications}} + \underbrace{xz}_{\text{1 multiplication}}$$

Algorithm Recursive Multiplication

function REC-MULTIPLY$(A, B, 2n)$ $\quad \triangleright\ n = 2^k$
    if $n = 1$ **then return** $A * B$
    else
        $A = 10^n w + x, \; B = 10^n y + z$
        $wy \leftarrow$ REC-MULTIPLY(w, y, n)
        $wz \leftarrow$ REC-MULTIPLY(w, z, n)
        $xy \leftarrow$ REC-MULTIPLY(x, y, n)
        $xz \leftarrow$ REC-MULTIPLY(x, z, n)
        return $10^{2n}(wy) + 10^n(wz + xy) + xz$

$$T(2n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n) + 6n & \text{if } n > 1 \end{cases}$$

$$= O(n^2)$$

Complexity

Multiplying two n digits integers

$$AB = (10^n w + x)(10^n y + z) = 10^{2n} \underbrace{(wy)}_{\text{1 multiplication}} + 10^n \underbrace{(wz + xy)}_{\text{2 multiplications}} + \underbrace{xz}_{\text{1 multiplication}}$$

$$\underline{wz + xy} = (w + x)(y + z) - wx - yz = wx + \underline{wz + xy} + xz - wy - xz$$

Algorithm Karatsuba Multiplication

---

function KARTASUBA$(x, y, 2n)$ $\quad\triangleright\ n = 2^k$
  if $n = 1$ **then return** $A * B$
  else
    $A = 10^n w + x,\ B = 10^n y + z$
    $wy \leftarrow$ KARTASUBA(w, y, n)
    $xz \leftarrow$ KARTASUBA(x, z, n)
    $mid \leftarrow$ KARTASUBA$(w + x, y + z, n)$
    return $10^{2n}(wy) + 10^n(mid - wy - xz) + xz$

$$T(2n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n) + 6n & \text{if } n > 1 \end{cases}$$

$$= O(n^{1.58})$$

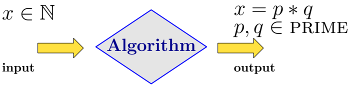
Complexity

Multiplying two n digits integers

- Repeated Addition (adding x to itself y times) ▷ $O(n10^n)$
- Long Multiplication ▷ $O(n^2)$
  - Kolmogorov (1960) conjecture: grade-school algorithm is the best possible
- Karatsuba's Algorithm (1960) ▷ $O(n^{1.58})$
- Harvey & van der Hoeven (2019) ▷ $O(n \log n)$
- Can we do better? ▷ Not known either way

Cryptography

Factorizing an n digits integer



▷ p and q large primes ($\sim n/2$-digits) $\implies$ $x = pq$ is n-digits long

Factorizing x into p and q is very hard ▷ Inverse of multiplication

1. Try all factors from 3 to x ▷ $O(10^n)$
2. Try all factors from 3 to $\sqrt{x}$ ▷ $O(10^{n/2})$
3. Use number field sieve ▷ $O(10^{n/3})$

▷ This is essentially the best known method

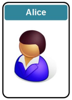
No "efficient algorithm" to find p and q from x ▷ widely believed

Indeed no "efficient algorithm" to check any non-trivial property of p or q

e.g. Is one of p and q have last digit 3?

Cryptography

Alice and Bob each want to win a coin flip over phone



- I will flip, you call it

- Ok, Heads!

- Sorry, it was tails

Cryptography

Alice and Bob each want to win a coin flip over phone

No efficient algorithm to check if p or q has last digit $= 3$ from $x(= pq)$

To call heads, choose p and q, so none ends with 3

To call tails, choose p and q, so at least one ends with 3

- **Sends** $x\ (= p * q)$
- **Sends** p **and** q

- **I will flip** 

- **Sends outcome of flip**
- **Checks if** $x = p * q$

Can Bob cheat? ▷ Can he guess last digit of p and q?

Can Alice cheat? ▷ Can she find $x = p' * q'$?

What if Alice choose p, q, r with r ending with 3 and reveals p and qr or pq and r?
▷ Primality testing is efficient