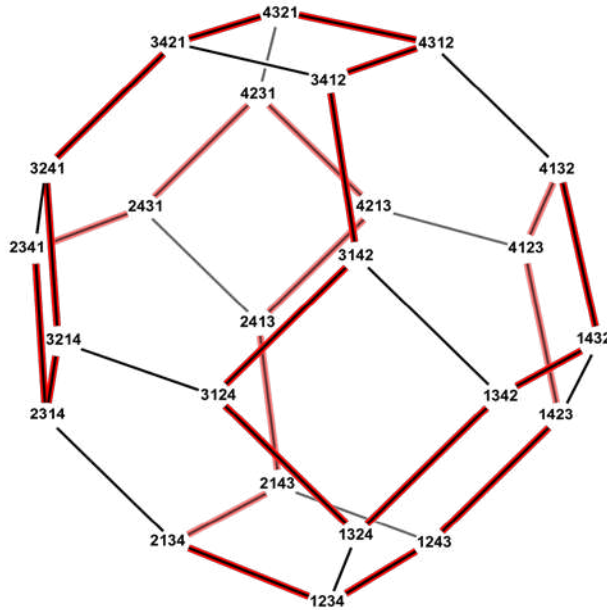


# CS3334 Data Structures

## Revision

(revisiting old problems with new perspectives)



Chee Wei Tan

# Motivation

- “Before there were **computers**, there were **algorithms**.”
  - Cormen, Leiserson, Rivest and Stein, Preface of Introduction to Algorithms
- Before there were **algorithms**, there were **abstract data types**. Now, what do you see in an **integer number**, a **permutation**, a **sequence**, a **recurrence** or a **graph**?
- “But now there are **computers**, there are even more **algorithms**, and **algorithms** lie at the heart of **computing**”
  - Cormen, Leiserson, Rivest and Stein, Preface of Introduction to Algorithms
- What **computing problem** intrigues you? And what **algorithms** have you discovered?

# Find the O-notation of a function

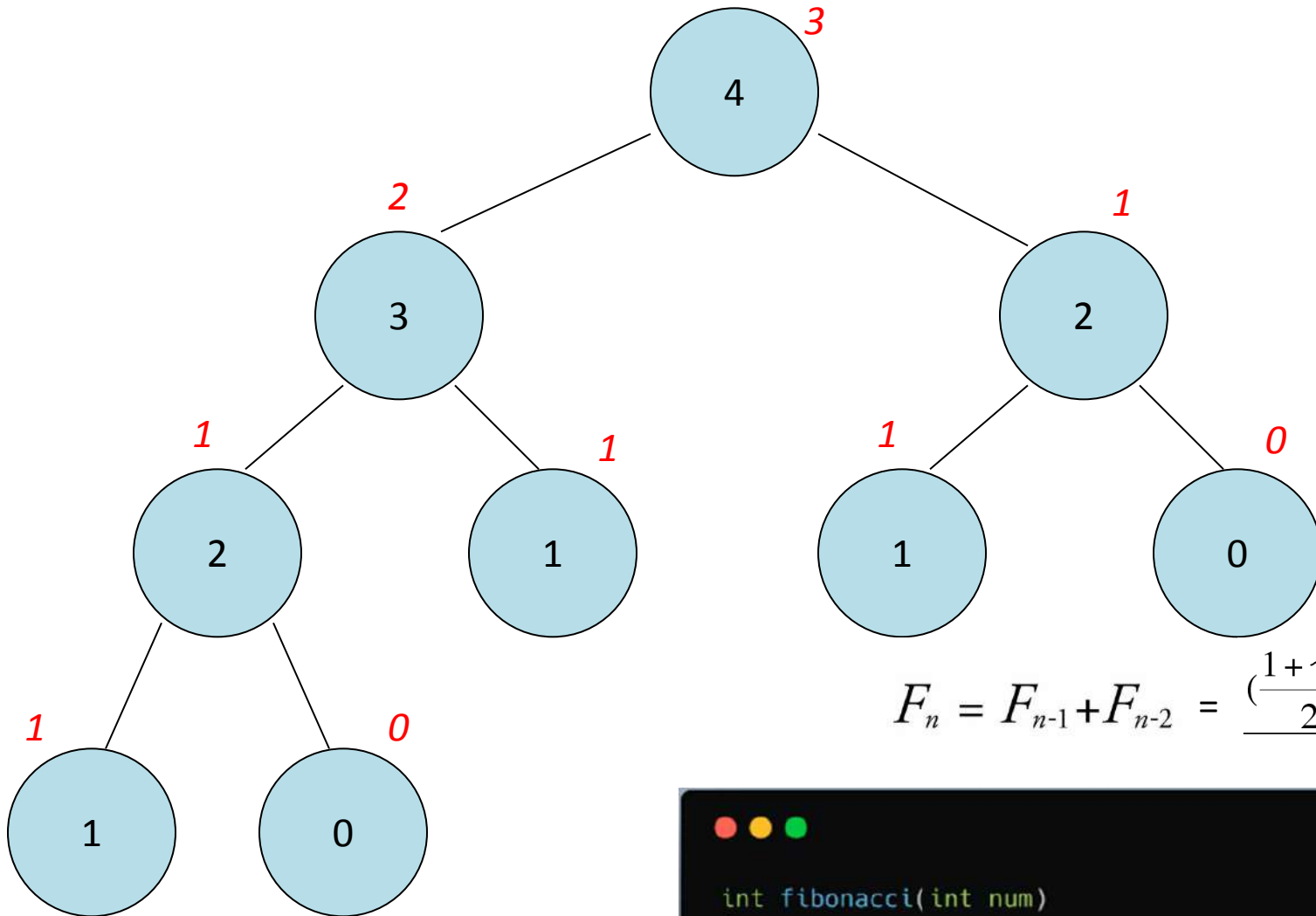
- Given a function with a **constant number of terms**, we can obtain the O-notation of the function by using the following three rules:
- **Rule #1:** Sort the terms of the function in decreasing order of their growth rates
- **Rule #2:** Ignore lower order terms (i.e., only keep the leading term)
- **Rule #3:** Ignore the coefficient of the leading term
- **Note:** These three rules DO NOT work for a function with a non-constant number of terms

# Fast Exponentiation and Recursion

- Compute  $x^n$ , given  $x$  and integer  $n \geq 0$
- Straightforward implementation requires  $O(n)$  time
  - $x^8 = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$
  - 7 multiplications,  $O(n-1) = O(n)$
- However, a recursive function can do it in  $O(\log n)$  time
  - It can compute  $x^8$  in 3 multiplications.
  - $(x) \cdot (x) = x^2$   
 $(x \cdot x) \cdot (x \cdot x) = x^4$   
 $(x \cdot x \cdot x \cdot x) \cdot (x \cdot x \cdot x \cdot x) = x^8$

```
double power(double x, int n)
{
    if (n==0) return 1; //base case
    if (n==1) return x; //base case
    if (n%2==0) //n is even
        return power(x*x,n/2); //else //n is odd
    return power(x*x,n/2)*x;
}
```

# Tree of Recursive Calls for Fibonacci Sequence



$$F_n = F_{n-1} + F_{n-2} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$


Return values are shown in *red italic*

Recursive Fu



```
int fibonacci(int num)
{
    if (num == 0 || num == 1)
        return num;
    else
        return fibonacci(num - 1) + fibonacci(num - 2);
}
```

# Recursive Binary Search



```
int binarySearch(int A[], int low, int up, int x)
{
    if (low>up) return -1; //cannot find x
    int mid = (low+up)/2;
    if (A[mid]==x) return mid; //find x
    else if (A[mid]<x)
        return binarySearch(A,mid+1,up,x);
    else
        return binarySearch(A,low,mid-1,x);
}
```

- Elements in A[] are sorted in increasing order.



Hi, here's a new puzzle for you!

Tell us what you know about this algorithm 😊 ? What if this recursive function takes as input 20180526 and 12345678? 😊 20180526 is the date of 2018 CS Challenge ! 😊😊

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

**What is its Tree of Recursion?**



A: 16

B: 8

C: 6

D: 12

Please select the correct answer by clicking one of the buttons below ⚡

<input type="radio"/>	A
<input type="radio"/>	B

<input type="radio"/>	C	>
<input type="radio"/>	D	



☰ Type a message...

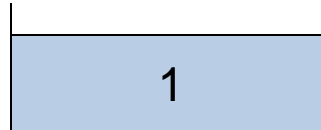


# Stacks: Example

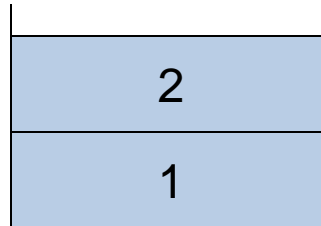
## Actions

1. Push(1)
2. Push(2)
3. Push(3)
4. Pop(x)
5. Pop(x)
6. Push(4)

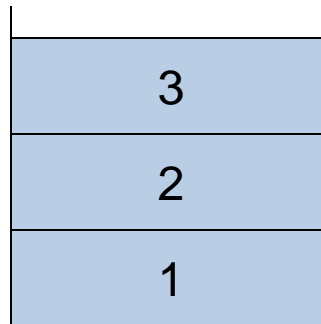
## Stack



1. Push(1)

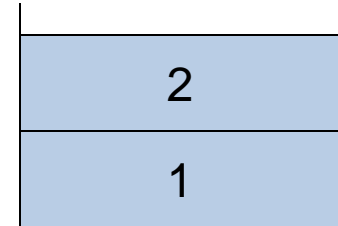


2. Push(2)

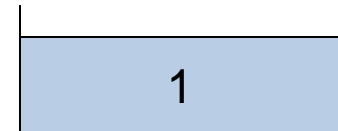


3. Push(3)

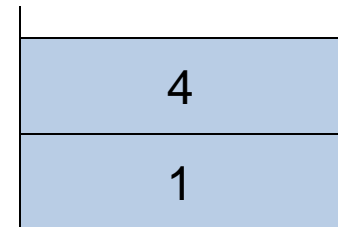
## Stack



4. Pop(x), x=3



5. Pop(x), x=2



6. Push(4)



# Tower of Hanoi: Sorting by Stacks

Carrier 2:18 PM

< Question 2 Question 3

How many moves will it take for three disks?


5

6

7

8

Restart Moves: 0



How long will it take for four disks?

How long will it take for  $n$  disks?

What is its Recurrence Equation?



# Quiz: Who invented the “Fibonacci Sequence” and “Tower of Hanoi”?

Carrier 2:18 PM

< Question 2 Question 3

How many moves will it take for three disks?


5

6

7

8

Restart Moves: 0



```
int fibonacci(int num)
{
    if (num == 0 || num == 1)
        return num;
    else
        return fibonacci(num - 1) + fibonacci(num - 2);
}
```

Can you write down a recurrence for the complexity of additions in fibonacci?



# Quiz: Who invented the “Fibonacci Sequence” and “Tower of Hanoi”?

Carrier 2:18 PM

< Question 2 Question 3

How many moves will it take for three disks?


5

6

7

8

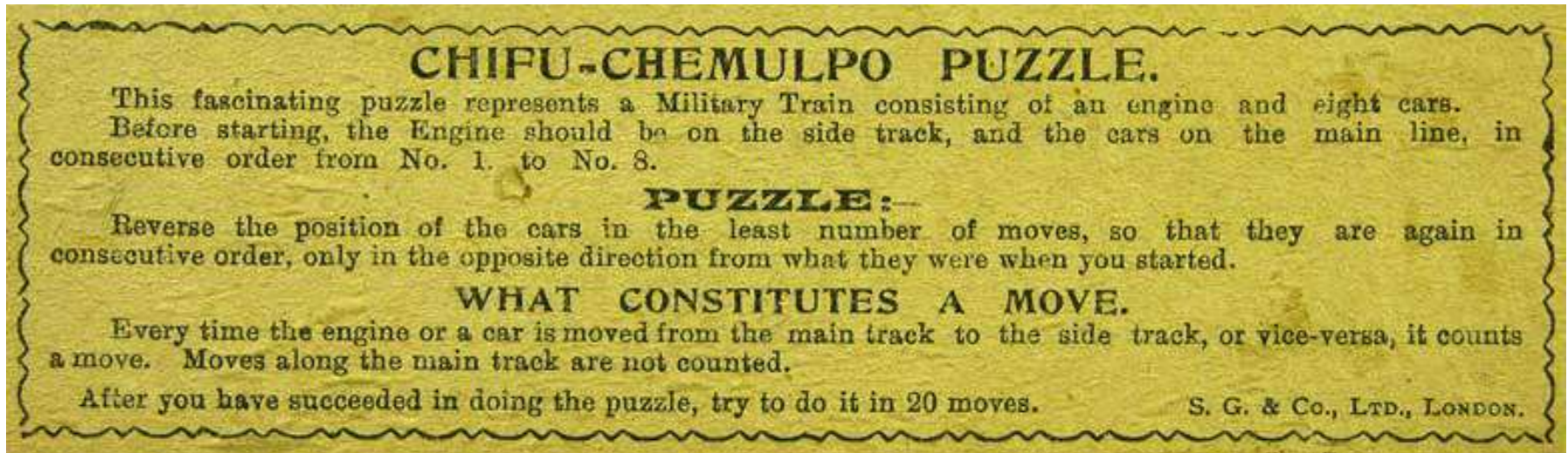
Restart Moves: 0



Édouard Lucas (1842-1891) was a French mathematician. In 1857, at age 15, Lucas began testing the primality of  $2^{127} - 1$  by hand, using [Lucas sequences](#). In 1876, after 19 years of testing, he finally proved that  $2^{127} - 1$  was prime; this would remain the largest known [Mersenne prime](#) for three-quarters of a century. The Tower of Hanoi puzzle appeared in 1883 under the name of M. Claus (Claus is an anagram of Lucas!) His four-volume work on recreational mathematics *Récréations mathématiques* is a classic.

[https://en.wikipedia.org/wiki/%C3%89douard\\_Lucas](https://en.wikipedia.org/wiki/%C3%89douard_Lucas)

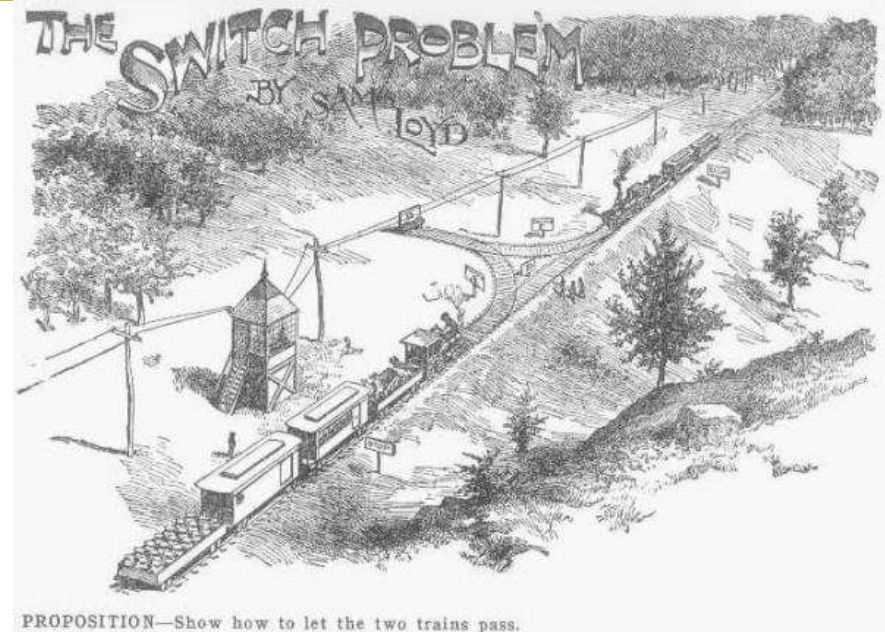
# Trains of Thought: Data Structures at Play



How to compute the solution with fewest moves for  $n$  cars ?



- 1) Wikipedia on Stacks and Queues  
[https://en.wikibooks.org/wiki/Data\\_Structures/Stacks\\_and\\_Queues](https://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queues)
- 2) A. J. T. Colin, A. D. McGettrick and P. D. Smith, Sorting Trains, The Computer Journal, Volume 23, 1978
- 3) Trains of Thought, B. Hayes, American Scientist, the magazine of Sigma Xi, The Scientific Research Society, Volume 95, 2007





# Sorting Since Time Immemorial



Plimpton 322 Tablet: Sorted Pythagorean Triples

<https://www.maa.org/sites/default/files/pdf/news/monthly105-120.pdf>



**Periodische Gesetzmässigkeit der Elemente nach Mendeleeff**

Reihen	Gruppe I R <sup>2</sup> D	Gruppe II R O	Gruppe III R <sup>2</sup> O <sup>3</sup>	Gruppe IV R H <sup>4</sup> R O <sup>2</sup>	Gruppe V R H <sup>3</sup> R <sup>2</sup> O <sup>5</sup>	Gruppe VI R H <sup>2</sup> R O <sup>3</sup>	Gruppe VII R H R <sup>2</sup> O <sup>7</sup>	Gruppe VIII R O <sup>4</sup>
1	H=1							
2	Li=7	Be=9,4	B=11	C=12	N=14	O=16	F=19	
3	Na=23	Mg=24	Al=27,3	Si=28	P=31	S=32	Cl=35,5	
4	K=39	Ca=40	Sc=44	Ti=48	V=51	Cr=52	Mn=55	Fe=56, Co=59 Ni=59, Cu=63
5	(Cu=63)	Zn=65	Ga=68	--72	As=75	Se=79	Br=80	
6	Rb=85	Sr=87	Yt=88	Zr=90	Nb=94	Mo=96	--100	Ru=104, Rh=104 Pd=106, Ag=108
7	(Ag=108)	Cd=112	In=113	Sn=118	Sb=122	Te=125	J=127	
8	Cs=133	Ba=137	Ce=137	La=139	-	Di=145?	-	- - -
9	(-)	-	-	-	-	-	-	- - -
10	-	165	169	Er=170	-173	Ta=182	W=184	-
11	(Au=196)	Hg=200	Tl=204	Pb=208	Bi=210	-	-	Pt=195, Os=195u Ir=193, Au=196
12	-	-	-	Th=231	-	U=240	-	- - -

Oldest Periodic Table of Elements: Sorted elements in order of increasing atomic number (atomic mass)

<https://www.bbc.com/bitesize/guides/z36cfcw/revision/1>

<https://www.theguardian.com/science/2019/jan/17/st-andrews-oldest-surviving-wall-chart-of-periodic-table-university>

# Sorting Since 1945



John Mauchly and J. Presper Eckert  
 - ENIAC (1946)  
 - Invented **Insertion Sort** (1946)  
 - UNIVAC - Information Age: Then and Now,  
 Computer History Museum, 1960  
<https://www.youtube.com/watch?v=h4wQJfdhOIU>

(1) A  $r+1$ -complex:  $X^{(p)}(x^0, x^1, \dots, x^r)$  consists of the main number:  $x^0$  and the satellites:  $x^1, \dots, x^r$ . Throughout what follows  $p = 1, 2, \dots$  will be fixed. A complex  $X^{(p)}$  precedes a complex  $Y^{(p)}$  if their main numbers are in this order:  $x^0 \leq y^0$ .

An  $n$ -sequence of complexes:  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_n}^{(p)}\}$ .

If  $\sigma_1, \dots, \sigma_n$  is a permutation of  $0, \dots, (n-1)$ , then the sequence  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_n}^{(p)}\}$  is a permutation of the sequence  $\{X_0^{(p)}, \dots, X_{n-1}^{(p)}\}$ .

A sequence  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_n}^{(p)}\}$  is monotone if its elements appear in their order of precedence:  $X_{\sigma_1}^{(p)} \leq X_{\sigma_2}^{(p)} \leq \dots \leq X_{\sigma_n}^{(p)}$ , i.e.  $x^0 \leq x^0 \leq \dots \leq x^0$ .

Every sequence  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_n}^{(p)}\}$  possesses a monotone permutation:  $\{X_{\sigma'_1}^{(p)}, \dots, X_{\sigma'_n}^{(p)}\}$  (at least one). Obtaining this monotone permutation is the operation of sorting the original sequence.

Given two (separately) monotone sequences  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_m}^{(p)}\}$  and  $\{Y_{\tau_1}^{(p)}, \dots, Y_{\tau_n}^{(p)}\}$  sorting the composite sequence  $\{X_{\sigma_1}^{(p)}, \dots, X_{\sigma_m}^{(p)}, Y_{\tau_1}^{(p)}, \dots, Y_{\tau_n}^{(p)}\}$  is the operation of meshing.

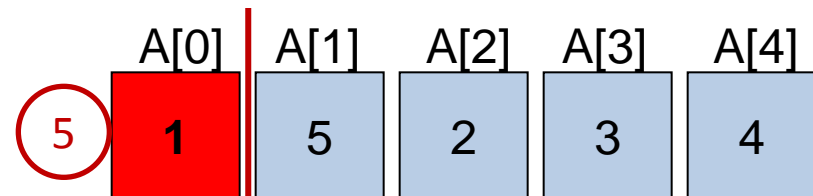
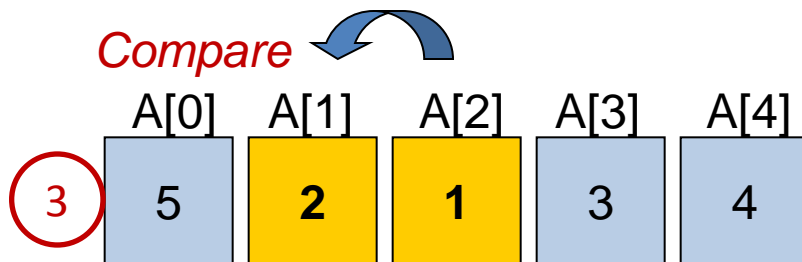
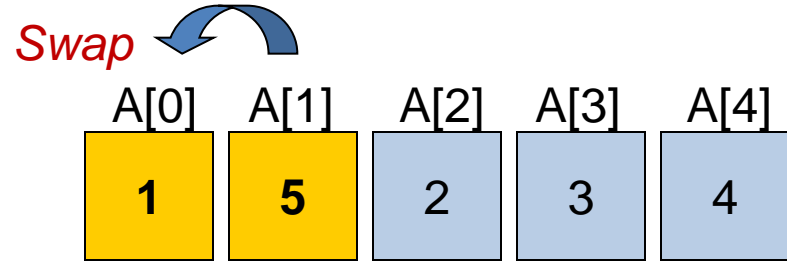
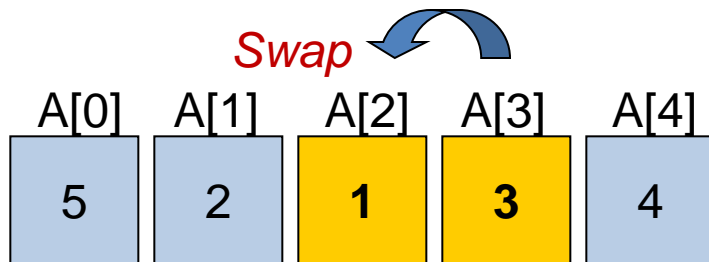
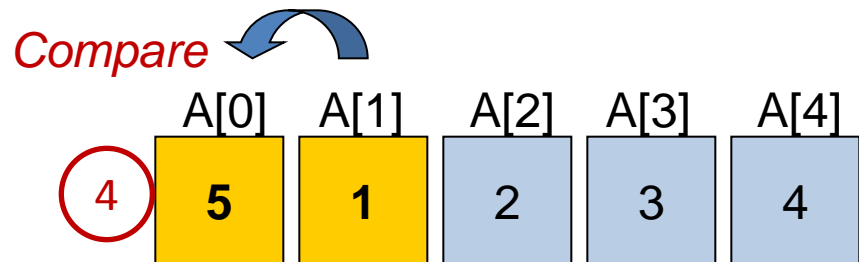
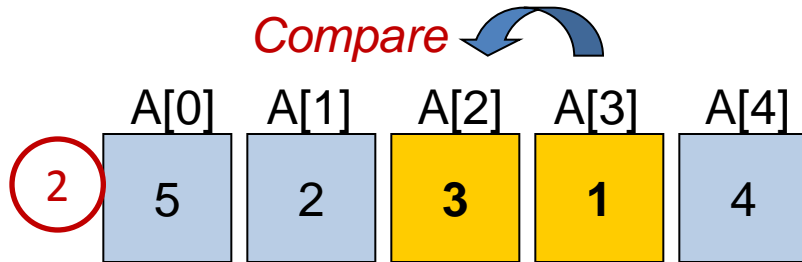
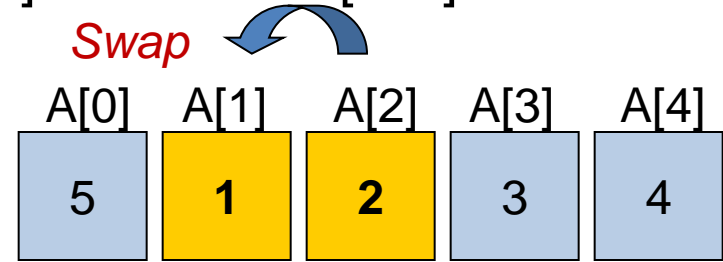
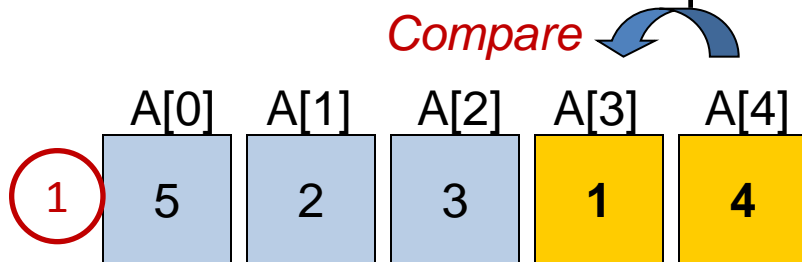
(2) We wish to formulate code instructions for sorting and for meshing, and to see how much control-capacity they tie up and how much time they require. It is convenient to consider meshing first and sorting afterwards.

First Stored Program <https://www.amphilsoc.org/exhibits/treasures/vonneuma.htm>

It is striking that von Neumann and Mauchly explain meshing in as much detail as its application to sorting, and highlight the use of the '**comparison operator**' to discriminate between two data items. The **automation of conditional control** was a new and untested technology in 1945, and although the meshing procedure might appear trivial to us, we should not underestimate its novelty. As Knuth (1973, 384) suggested, implementing these **non-numerical procedures** did provide reassurance that, as John von Neumann (1945e) put it, "**the present principles for the logical controls are sound**". - Routines of Substitution, John von Neumann's Work on Software Development, 1945–1948, Mark Priestley (2018)

# Bubble Sort

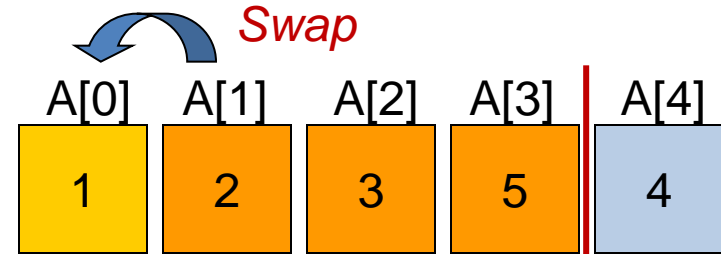
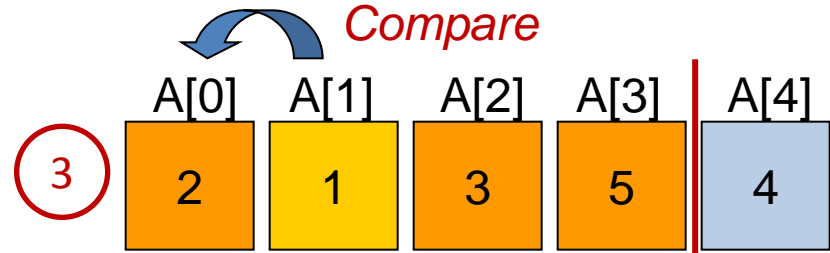
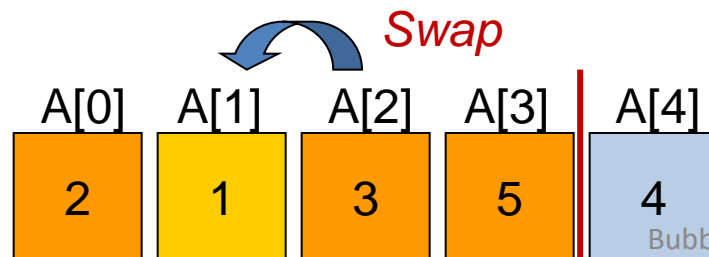
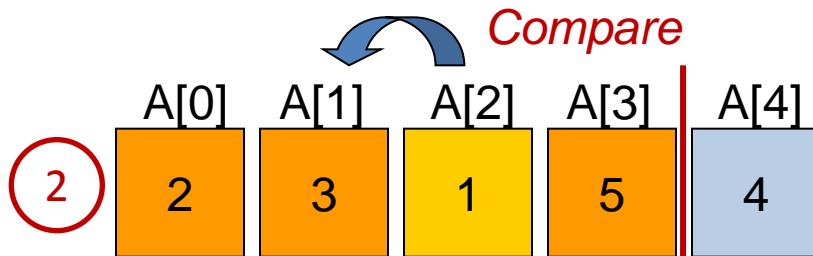
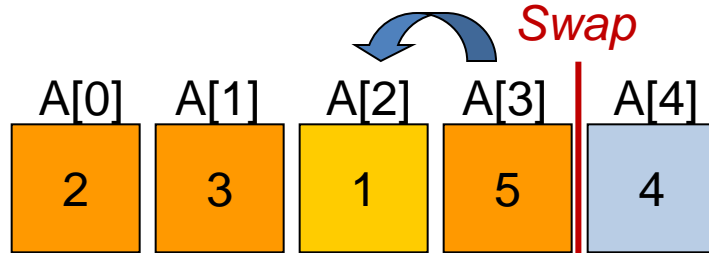
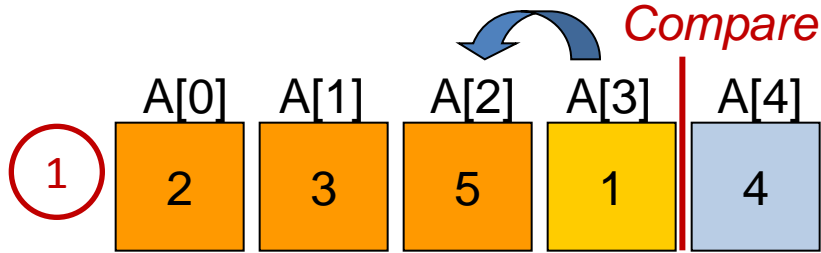
Just before pass 1:  $A[0..-1]$  sorted &  $\leq A[0..4]$



After pass 1: smallest of  $A[0..4]$  at  $A[0]$   
 $A[0..0]$  sorted and  $\leq A[1..4]$

# Insertion Sort

Just before pass 3:  $A[0..2]$  is sorted version of old  $A[0..2]$

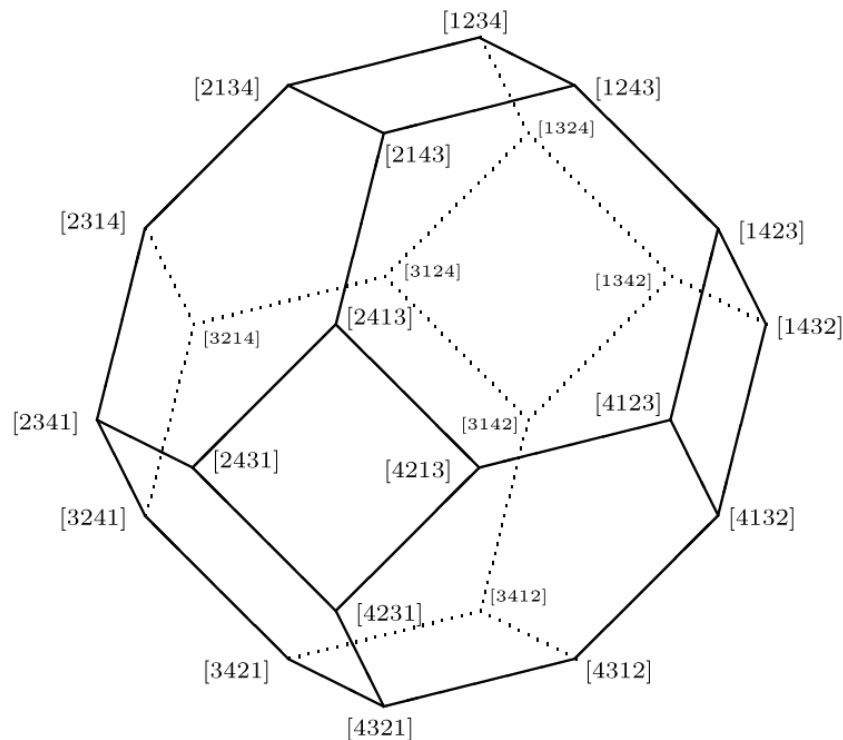


After pass 3:  $A[0..3]$  is sorted version of old  $A[0..3]$   
Pass 4 will consider  $A[4]$



# Geometry of Inversions

- Example of permutation of four elements.
- Visualizes as a semi-octahedron where the number of inversions of a particular permutation is the length of a downward path from  $[1234]$  to that permutation.



# How to Solve It?



*George Pólya*

- 1) First, you have to understand the problem.
- 2) After understanding, make a plan.
- 3) Carry out the plan.
- 4) Look back on your work. How could it be better?

If this fails, Pólya advises: "If you can't solve a problem, then there is an easier problem you can solve: find it."

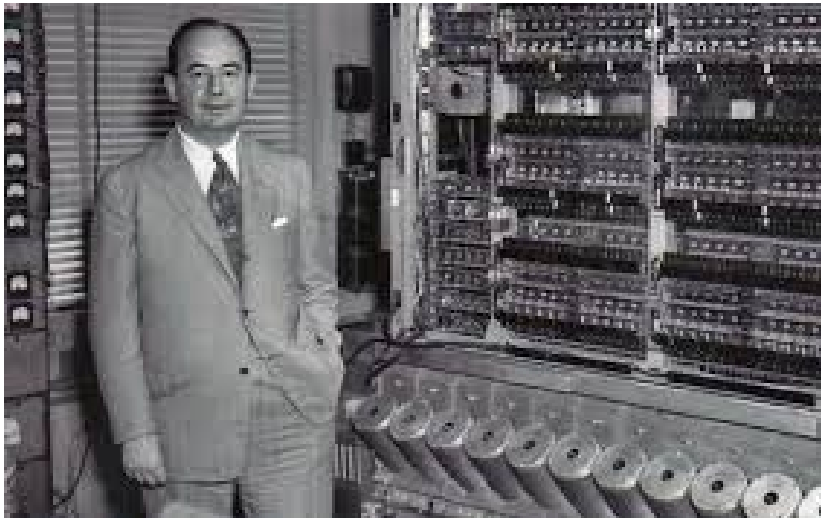
[https://en.wikipedia.org/wiki/How to Solve It](https://en.wikipedia.org/wiki/How_to_Solve_It)

Clearly expound the idea of **Divide-and-Conquer**

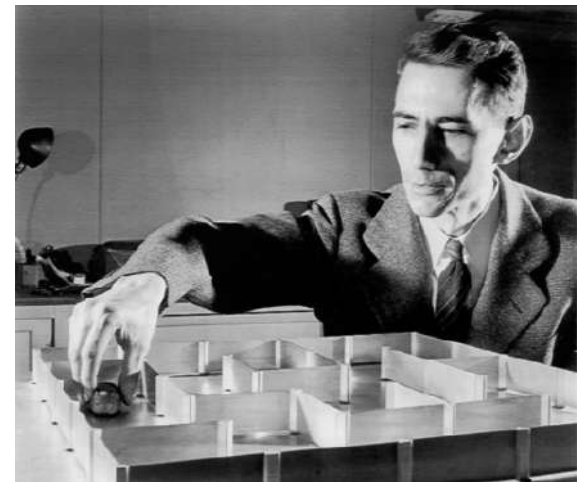
Break down a problem into smaller ones you can handle/play with

"Johnny (von Neumann) was the only student I was ever afraid of," *George Pólya*

# Giants of Computer Science



*If you come up with a clever algorithm that's clean and elegant, you have a much better chance of people using it.*



# Idea: Divide and Conquer

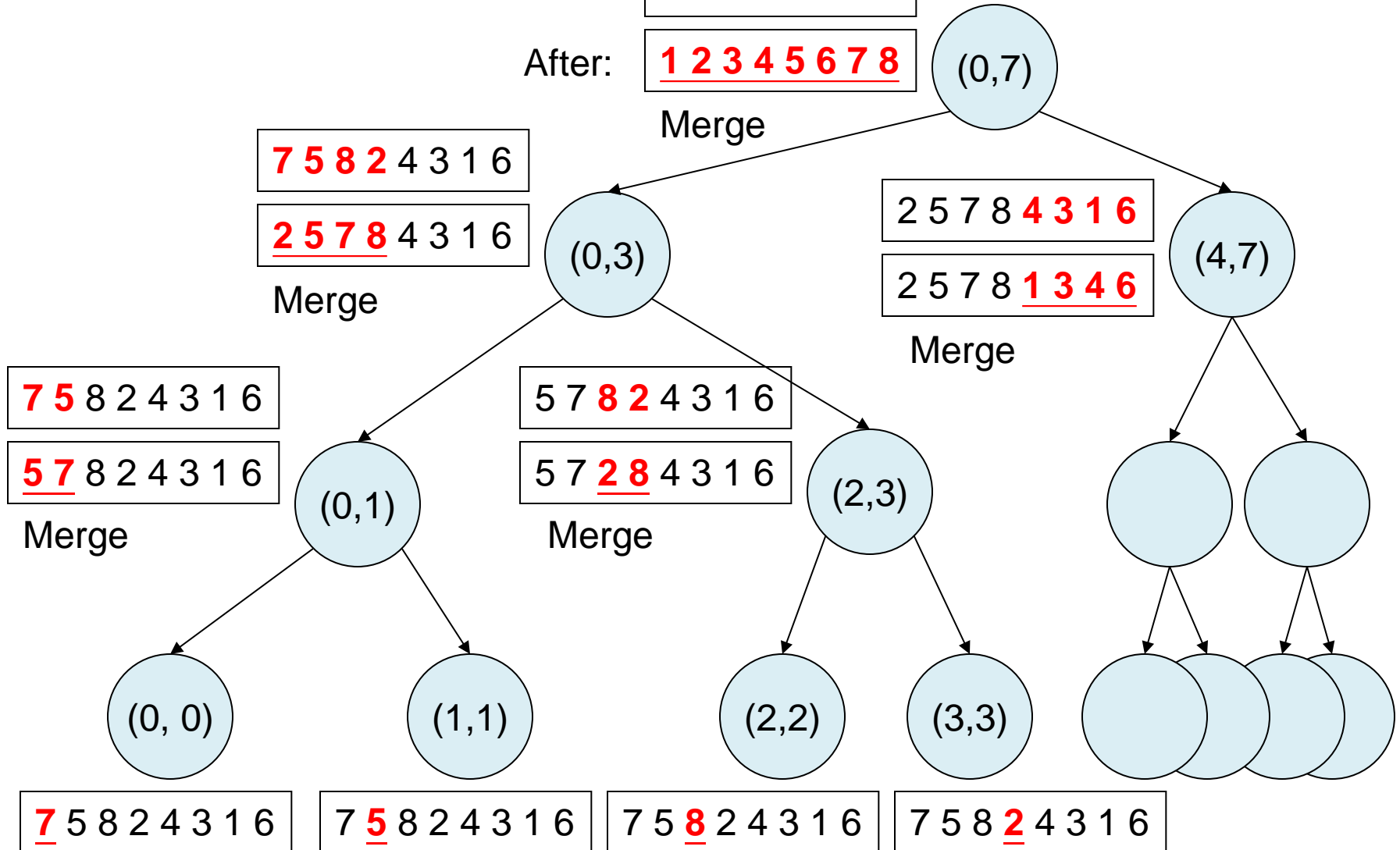
- **Divide:** Divide a problem into smaller and simpler subproblems
- **Conquer:** Solve recursively each subproblem
- **Merge:** Combine solutions to subproblems to get the solution to original problem
- Examples of historical algorithms:
  - Gauss Fast Fourier Transform (1805)
    - Gauss discovered the algorithm at age 28
  - Merge Sort algorithm (1945)
    - John von Neumann may have discovered it by playing poker
  - Karatsuba algorithm (1960)
    - Karatsuba discovered the algorithm at age 23



# Merge Sort

Before: 7 5 8 2 4 3 1 6

After: **1 2 3 4 5 6 7 8**



# Merge Sort Theorem

Suppose  $n$  is a power of 2. Consider the recurrence relation of the form

$$T(n) = 2 T(n/2) + n \quad \text{if } n > 1 \text{ and } T(1) = 0.$$

Then  $T(n) = n \log_2 n$ .

[Quiz] Give a proof by induction.

- Theorem: Any *comparison-based* sorting algorithm requires  $\Omega(n \log n)$  time
- Examples of comparison-based sorting algorithms:
  - Bubble Sort, Insertion Sort:  $O(n^2)$  time
  - Merge Sort, Heap Sort:  $O(n \log n)$  time
  - Quick Sort:  $O(n^2)$  worst case,  $O(n \log n)$  average case



# Quick Sort is Optimal



**Sir Charles Antony Richard Hoare**  
Inventor of Quick Sort in 1959



**Prof. Robert Sedgewick**  
His influential Ph.D. thesis in 1975  
resolved open issues in Quick Sort

## QUICKSORT

by Robert Sedgewick

### Abstract

A complete study is presented of the best general purpose method for sorting by computer: C. A. R. Hoare's Quicksort algorithm. Special attention is paid to the methods of mathematical analysis which are used to demonstrate the practical utility of the algorithm. The most efficient known form of Quicksort is developed, and exact formulas are derived for the average, best case, and worst case running times. The merits of the many modifications which have been suggested to improve Quicksort are discussed, with an emphasis on their impact upon the analysis. Van Emden's method, samplesort, and the median-of-three modification are discussed in detail, and it is shown that the latter is the most effective improvement to Quicksort for practical sorting applications.

New results presented include: improvements to the algorithm based on a refined partitioning strategy and a new method of handling small subfiles, the best and worst case analysis, contrasting analyses of minor variants and the study of the effect of equal keys, new implementations of and new approaches to analyzing adaptive partitioning and samplesort, the complete general analysis of fixed sample size partitioning, and the application of "loop unwrapping" to Quicksort and the analysis of the optimized program.

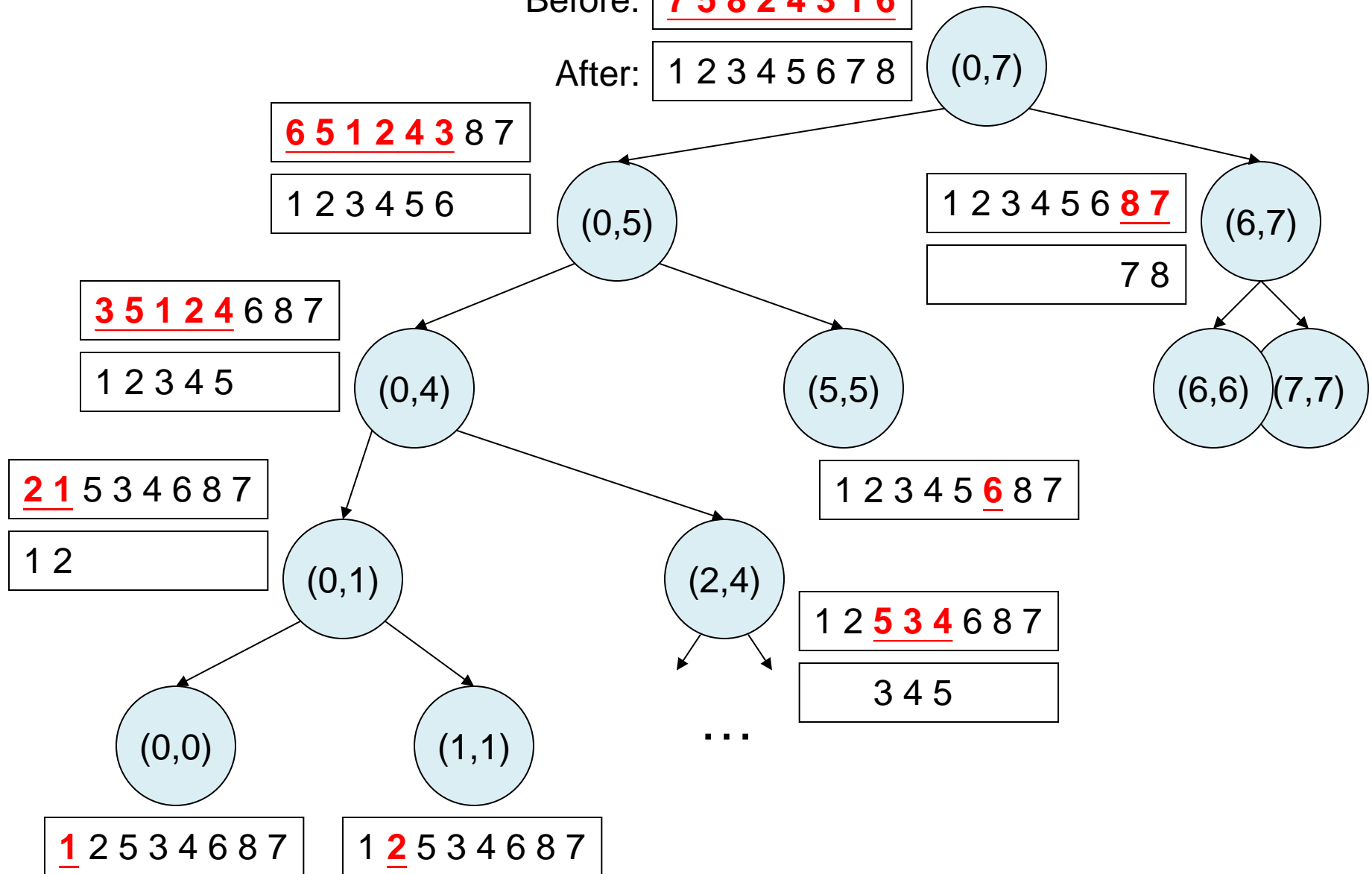
The thesis is presented in an expository fashion so that it may be useful as a textbook in the field of "analysis of algorithms". It is self-contained, and it includes a complete treatment of a simpler sorting algorithm (insertion sorting) as well as three appendices which complement the material in the text.

This research was supported in part by The Fannie and John Hertz Foundation. The printing of this paper was supported in part by National Science Foundation grant GJ 36473X and by the Office of Naval Research contract NR 044-402.

# Quick Sort

Before: 7 5 8 2 4 3 1 6

After: 1 2 3 4 5 6 7 8

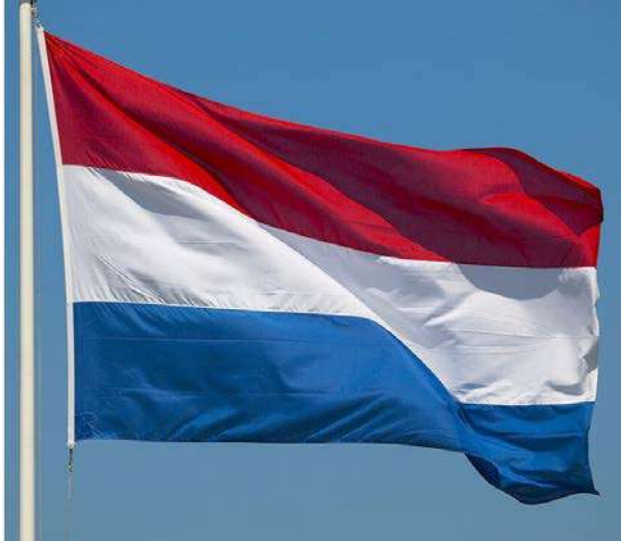




# Average Case Analysis

- Same model as in Lec 4:
  - Assume input array is a permutation of  $\{1, \dots, n\}$
  - For a fixed  $n$ , there are  $n!$  possible inputs
  - “Average case time complexity” means “average running time of these  $n!$  inputs”
- Let  $T_A(n)$  = average case time complexity of quick sort
- Theorem:  $T_A(n) = O(n \log n)$
- Advanced proof involving concepts of entropy and probability (covered in more advanced algorithm courses).

# The Dutch National Flag Problem



The [flag of the Netherlands](#) consists of three colors: red, white and blue. Given balls of these three colors arranged randomly in a line (the actual number of balls does not matter), arrange the balls such that all balls of the same color are together and their collective color groups are in the correct order.

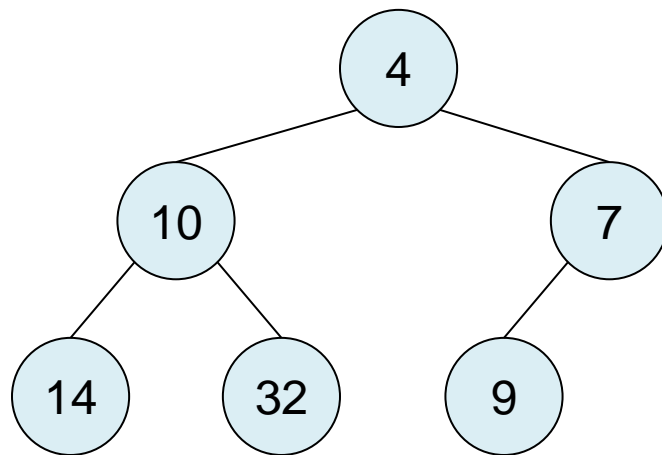
## Edsger W. Dijkstra

One of the most influential figures of computing science's founding generation, Dijkstra was a theoretical physicist whose career was a computer programmer. His ideas lay the foundations for the birth and development of the professional discipline of software engineering. And he gave his name to one of the most famous algorithms in graph theory.



# Heap Sort

- Heap Sort
  - Build a min-heap on the  $n$  elements, i.e.,  $O(n)$  time
  - Repeat to remove the minimum element  $n$  times, i.e.,  $O(n \cdot \log n)$  time
- Complexity:  $O(n + n \cdot \log n) = O(n \log n)$



# Bucket Sort (Example)

- Input: 3, 4, 6, 9, 4, 3 where  $M=10$

Counter array A:

0	1	2	3	4	5	6	7	8	9

- Step 1: Initialization

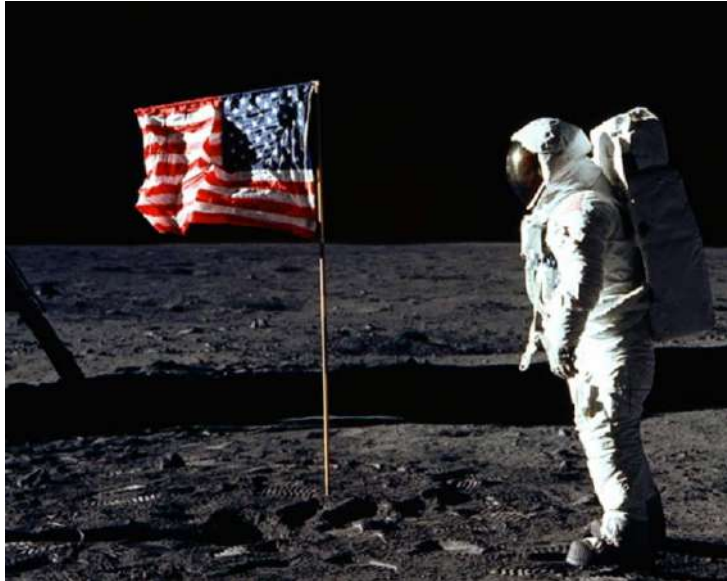
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

- Step 2: Read 3

$(A[3] = A[3] + 1)$

0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	0	0	0	0

# The American Flag Sorting Problem



The name comes by analogy with the Dutch national flag problem (see previous lecture): efficiently partition the array into many "stripes".

## Engineering Radix Sort

*Peter M. McIlroy*

*Keith Bostic*

Computer Science Research Group  
University of California at Berkeley

*M. Douglas McIlroy*

AT&T Bell Laboratories

### ABSTRACT

Radix sorting methods have excellent asymptotic performance on string data, for which comparison is not a unit-time operation. Attractive for use in large byte-addressable memories, these methods have nevertheless long been eclipsed by more easily programmed algorithms. Three ways to sort strings by bytes left to right—a stable list sort, a stable two-array sort, and an in-place “American flag” sort—are illustrated with practical C programs. For heavy-duty sorting, all three perform comparably, usually running at least twice as fast as a good quicksort. We recommend American flag sort for general use.

McIlroy, Peter M.; Bostic, Keith; McIlroy, M. Douglas (1993). "[Engineering radix sort](#)". *Computing Systems*. 6 (1): 5–27.

### 1. Introduction

For sorting strings you can't beat radix sort—or so the theory says. The idea is simple. Deal the strings into piles by their first letters. One pile gets all the empty strings. The next gets all the strings that begin with *A*–; another gets *B*– strings, and so on. Split these piles recursively on second and further letters until the strings end. When there are no more piles to split, pick up all the piles in order. The strings are sorted.

In theory radix sort is perfectly efficient. It looks at just enough letters in each string to distinguish it from all the rest. There is no way to inspect fewer letters and still be sure that the strings are properly sorted. But this theory doesn't tell the whole story: it's hard to keep track of the piles.

Our main concern is bookkeeping, which can make or break radix sorting as a practical method. The paper may be read as a thorough answer to exercises posed in Knuth chapters 5.2 and 5.2.5, where the general plan is laid out.<sup>1</sup> Knuth also describes the other classical sorting methods that we refer to: radix exchange, quicksort, insertion sort, Shell sort, and little-endian radix sort.

# Hash Table – Query/Delete

Query(17): Yes

0	21
1	
2	9
3	17
4	
5	
6	

$$h(17) = 17 \bmod 7 = 3$$

Query(27): No

0	21
1	
2	9
3	17
4	
5	
6	

$$h(27) = 27 \bmod 7 = 6$$

Delete (9):

0	21
1	
2	
3	17
4	
5	
6	

$$h(9) = 9 \bmod 7 = 2$$

# Birthday Paradox in Hashing

- Use  $1 - x \leq e^{-x}$ , for all  $x$
- Therefore,  $1 - j \cdot p \leq e^{-jp}$
- Also,  $e^x + e^y = e^{x+y}$
- Therefore,  $P_k \leq e^{(-p - 2p - 3p \dots - (k-1)p)}$
- $P_k \leq e^{-k(k-1)p/2}$
- For  $k = 23$ , we have  $k(k-1)/2 \cdot 365 = 0.69$
- $e^{-0.69} \leq 0.4999$

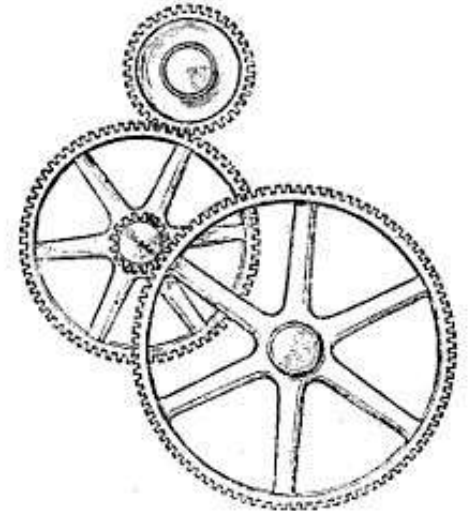
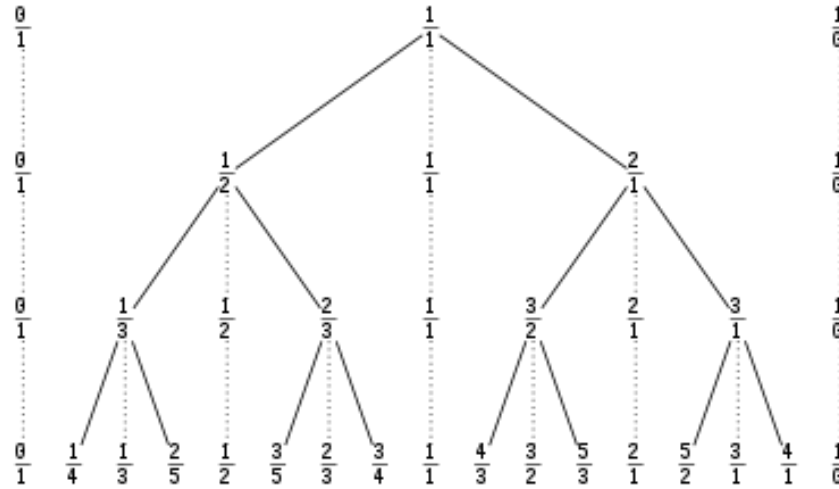
- Connection to Hashing:
  - Suppose  $n = 23$ , and hash table has size  $M = 365$ .
  - 50% chance that 2 keys will land in the same bucket
  - Collision

Insert(24):

0	21	
1		
2	9	
3	17	← collision
4		
5		
6		

$h(24) = 24 \bmod 7 = 3$

# Mathematics and Clockwork



In [number theory](#), the **Stern–Brocot tree** is an [infinite complete binary tree](#) in which the [vertices](#) correspond [one-for-one](#) to the [positive rational numbers](#), whose values are ordered from the left to the right as in a [search tree](#). The Stern–Brocot tree was discovered independently by **Moritz Stern** (1858) and **Achille Brocot** (1861). Stern was a German number theorist; Brocot was a French clockmaker who used the Stern–Brocot tree to design systems of gears with an engineered gear ratio.

[https://en.wikipedia.org/wiki/Stern%E2%80%93Brocot\\_tree](https://en.wikipedia.org/wiki/Stern%E2%80%93Brocot_tree)

This is a binary search tree data structure useful for approximating real numbers by rational numbers. What problems can you solve with it?





# Binary Search Tree

INFORMATION AND CONTROL 3, 327-334 (1960)

## Some Combinatorial Properties of Certain Trees With Applications to Searching and Sorting\*

THOMAS N. HIBBARD

*System Development Corporation, Santa Monica, California*

### INTRODUCTION

This paper introduces an abstract entity, the binary search tree, and exhibits some of its properties. The properties exhibited are relevant to processes occurring in stored program computers—in particular, to search processes. The discussion of this relevance is deferred until Section 2.

Section 1 constitutes the body of the paper. Section 1.1 consists of some mathematical formulations which arise in a natural way from the somewhat less formal considerations of Section 2.1. The main results are Theorem 1 (Section 1.2) and Theorem 2 (Section 1.3).

The initial motivation of the paper was an actual computer programming problem. This problem was the need for a list which could be searched efficiently and also changed efficiently. Section 2.1 contains a description of this problem and explains the relevance to its solution of the results of Section 1.

Section 2.2 contains an application to sorting.

The reader who is interested in the programming applications of the results but not in their mathematical content can profit by reading Section 2 and making only those few references to Section 1 which he finds necessary.

### 1. COMBINATORIAL PROPERTIES OF BINARY SEARCH TREES

#### 1.1 Preliminary Definitions

The central notion of this paper is that of a certain type of directed graph undergoing "random" operations. The present section is given to defining the graph and certain quantities associated with the graph. "Expected" values of these quantities will be defined combinatorially in Section 1.2; these "expected" values will then be calculated assuming "random insertions" (Section 1.2) and "random deletions" (Section 1.3).

**DEFINITION.** A *binary search tree* is a directed graph<sup>1</sup> having the following properties.

\* Received March, 1961; revised July, 1961.

## On the Efficiency of a New Method of Dictionary Construction

A. D. BOOTH AND A. J. T. COLIN

*Department of Numerical Automation, Birkbeck College, University of London*

Programs for constructing dictionaries of texts, with computers, have sometimes been adaptations of methods suitable for manual construction with card indexes. With all card index methods it is customary to keep the different words collected in alphabetical order, for the structure of a card index lends itself to such a process: all that is necessary to insert a new word into the index between two existing ones is to make out a new card and to put it in the correct place. However, the insertion of a new word in the store of a computer where the words are kept in alphabetical order is a time-consuming process, for all the words below the one which is inserted have to be "moved down" by one place.

If, however, a computer method is used where the words are not stored in alphabetical order but in the order in which they occur, the position is even worse, for although the shifting of words is eliminated, the dictionary search which is necessary for each word in the text, to establish whether it is a new word or not, must involve all the words collected so far, and not just a small number of them, as would be the case if the words were in alphabetical order, and a logarithmic search (Booth, 1955) could be used.

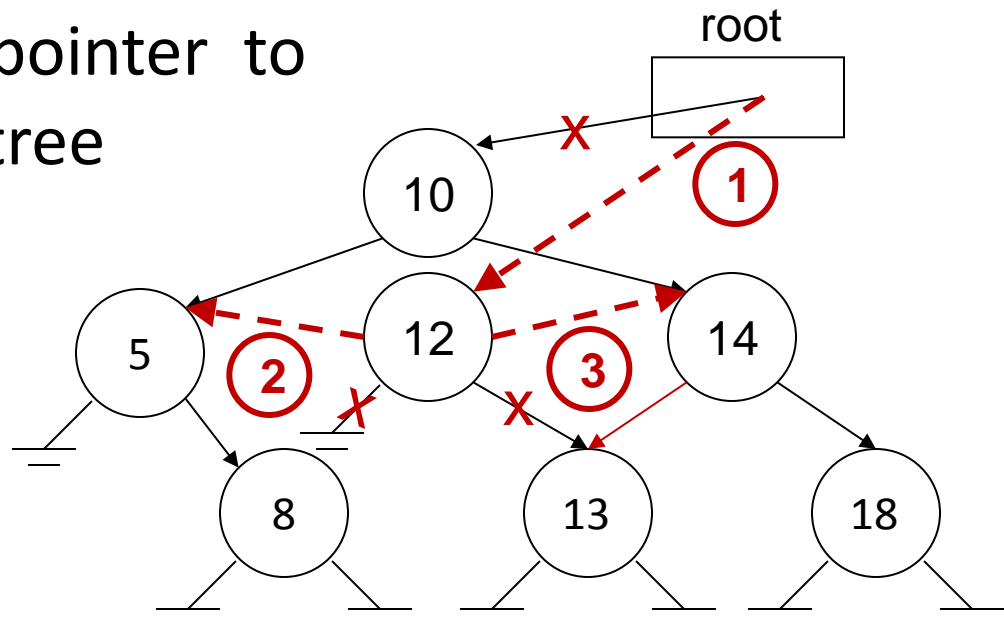
The method of construction described in this paper overcomes both these problems. It is not an adaptation of a manual method, but is designed specifically for computers. The method is based on a tree structure, which is discussed below.

### THE LOGICAL TREE

The logical tree is based on the fact that if  $\alpha$  and  $\beta$  are two different words,  $\alpha$  is either alphabetically less or alphabetically greater than  $\beta$ . The construction of a tree is illustrated by taking a sentence and making a tree from it.

# Binary Search Tree

- Example of Remove(10)
- Link 12 in place of 10
  1. Set pointer to 10 to 12
  2. Set 12's left pointer to 10's left subtree
  3. Set 12's right pointer to 10's right subtree



# AVL Trees in Computer Science

Received 22/MAR/62

## BIBLIOGRAPHY

- [1] C. Miranda, *Equazioni alle derivate parziali di tipo ellittico*, Springer, Berlin, 1955.
- [2] S. M. Nikol'skii, *Sibirsk. Mat. Ž.* 1 (1960), 78.

Translated by:  
F. M. Goodspeed

## AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

G. M. ADEL'SON-VEL'SKIĬ AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

**Statement of the problem.** The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in  $C \lg N$  operations, where  $N$  is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.

The reference board is a dyadic tree (Figure 1a): each of its cells has no more than one left cell, and no more than one right cell subordinate to it. Direct subordination induces subordination (partial ordering). In addition, for each cell of the tree, all the cells which are subordinate to a left (right) directly subordinate cell, will be arranged further to the left (right) than the given cell. Moreover, we assume that there is a cell (the head) to which all the others are subordinate. By transitivity, the conception "further to the left" and "further to the right" extends to the aggregate of all the cell pairs, and this aggregate becomes ordered. Thus, a given order of cells in a reference board should coincide with the order of arrangement of the estimates of the corresponding information elements (to be specific, we shall consider the estimates as increasing from left to right).

In the first address of each cell of the reference board, a place is indicated where the corresponding information element is located. The addresses of the cells of the reference board, which are directly subordinate on the left and right respectively to the given cell, are located in the second and third addresses. If a cell has no directly subordinate cells on either side, then there is zero in the corresponding address. The head address is stored in a certain fixed cell  $l$ .

Let us call the sequence of the cells of the tree a *chain* in which each previous cell is directly



Georgy Adelson-  
Velsky



Evgenii  
Landis

1962 paper "An algorithm for the organization of information" originally in Russian

# AVL Tree Theorem

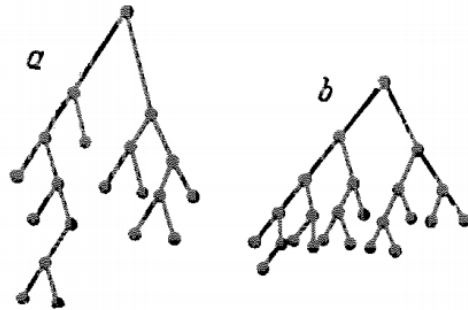


Figure 1

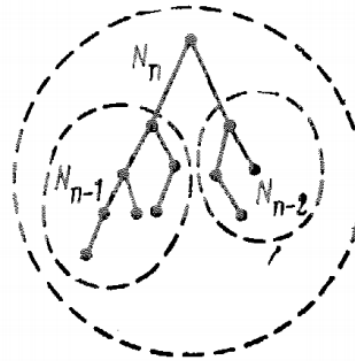


Figure 2

The recording algorithm is such that at each moment, the reference board is an admissible tree.

**Lemma 1.** *Let the number of cells of the admissible tree be equal to  $N$ . Then the maximum length of the branch is not greater than  $(3/2) \log_2 (N + 1)$ .*

**Proof.** Let us denote by  $N_n$  the minimum number of cells in the admissible tree when the given maximum length of the branch is  $n$ . Then it can be easily proven (see Figure 2) that  $N_n = N_{n-1} + N_{n-2} + 1$ .

When we solve this equation in finite remainders, we get

$$N_n = \left(1 + \frac{2}{\sqrt{5}}\right) \left(\frac{1 + \sqrt{5}}{2}\right)^n + \left(1 - \frac{2}{\sqrt{5}}\right) \left(\frac{1 - \sqrt{5}}{2}\right)^n - 1.$$

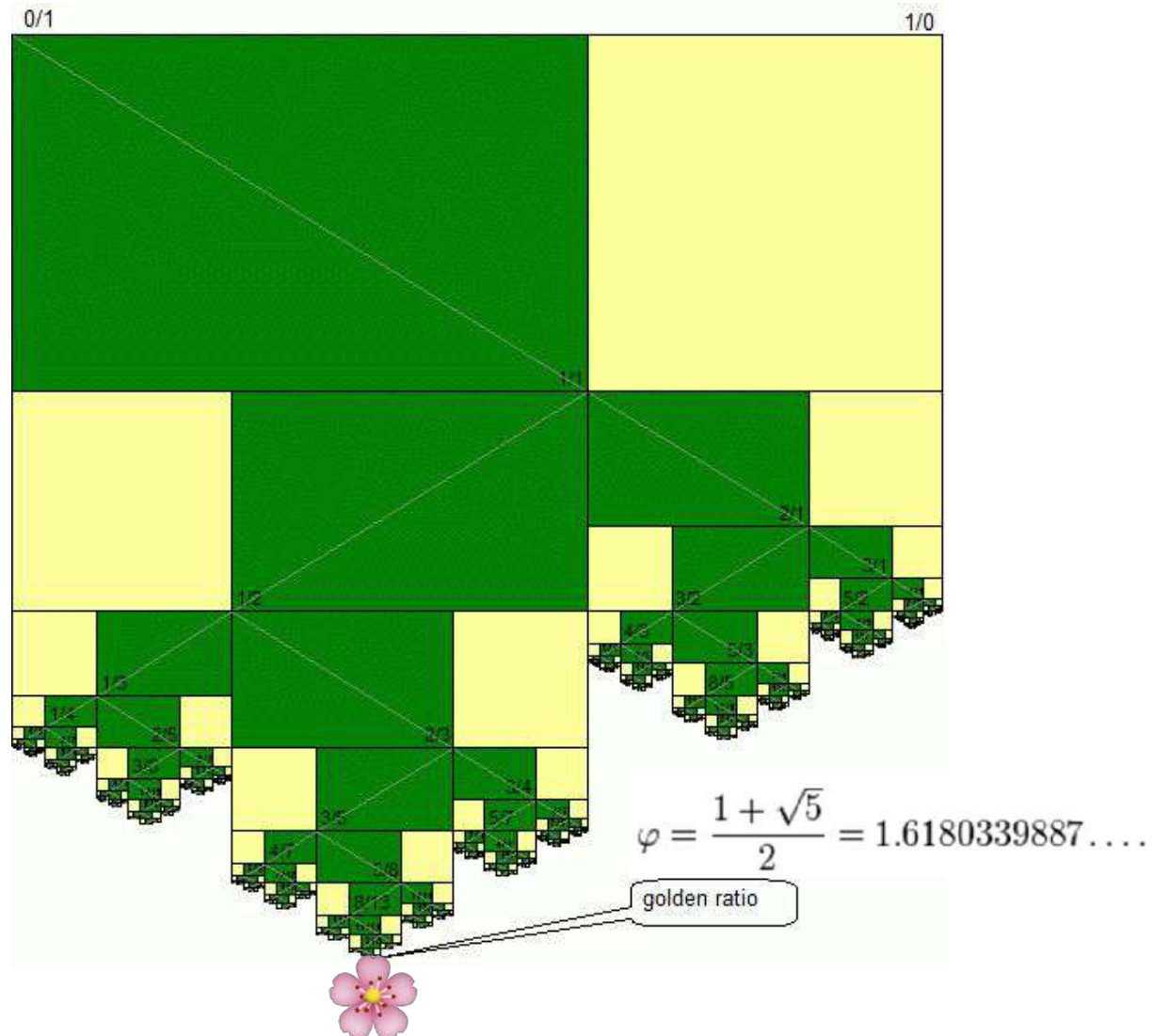
Whence

$$n < \log_{\frac{1 + \sqrt{5}}{2}} (N + 1) < \frac{3}{2} \log_2 (N + 1),$$

q.e.d.

Recall the NIM Game  
we played once before

# A Pretty Flower in Stern-Brocot Tree



# Kruskal's Algorithm

## ON THE SHORTEST SPANNING SUBTREE OF A GRAPH AND THE TRAVELING SALESMAN PROBLEM

JOSEPH B. KRUSKAL, JR.

Several years ago a typewritten translation (of obscure origin) of [1] raised some interest. This paper is devoted to the following theorem: If a (finite) connected graph has a positive real number attached to each edge (the *length* of the edge), and if these lengths are all distinct, then among the spanning<sup>1</sup> trees (German: Gerüst) of the graph there is only one, the sum of whose edges is a minimum; that is, the shortest spanning tree of the graph is unique. (Actually in [1] this theorem is stated and proved in terms of the "matrix of lengths" of the graph, that is, the matrix  $\|a_{ij}\|$  where  $a_{ij}$  is the length of the edge connecting vertices  $i$  and  $j$ . Of course, it is assumed that  $a_{ij}=a_{ji}$  and that  $a_{ii}=0$  for all  $i$  and  $j$ .)

The proof in [1] is based on a not unreasonable method of constructing a spanning subtree of minimum length. It is in this construction that the interest largely lies, for it is a solution to a problem (Problem 1 below) which on the surface is closely related to one version (Problem 2 below) of the well-known traveling salesman problem.

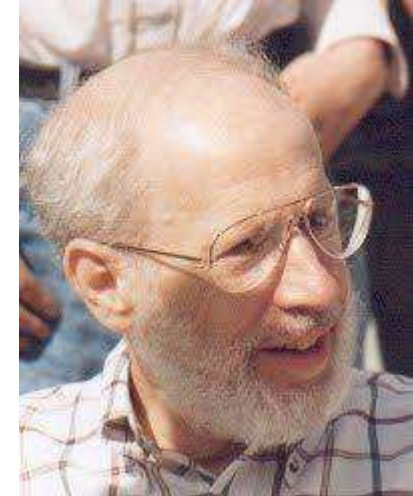
**PROBLEM 1.** Give a practical method for constructing a spanning subtree of minimum length.

**PROBLEM 2.** Give a practical method for constructing an unbranched spanning subtree of minimum length.

The construction given in [1] is unnecessarily elaborate. In the present paper I give several simpler constructions which solve Problem 1, and I show how one of these constructions may be used to prove the theorem of [1]. Probably it is true that any construction

Received by the editors April 11, 1955.

<sup>1</sup> A subgraph spans a graph if it contains all the vertices of the graph.



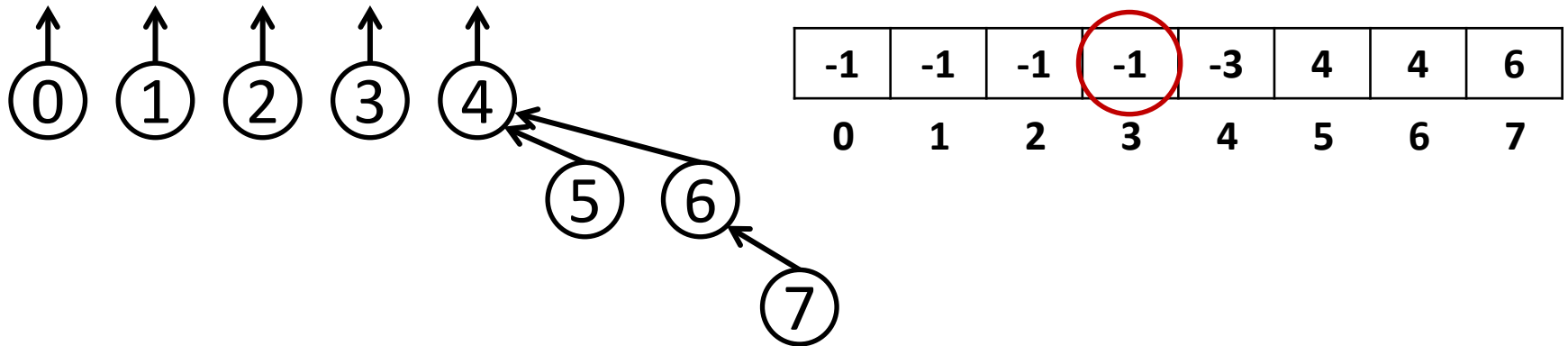
**Joseph Bernard Kruskal, Jr.** (1928 –2010) was an American mathematician, statistician, computer scientist and psychometrician.

[https://en.wikipedia.org/wiki/Joseph\\_Kruskal](https://en.wikipedia.org/wiki/Joseph_Kruskal)

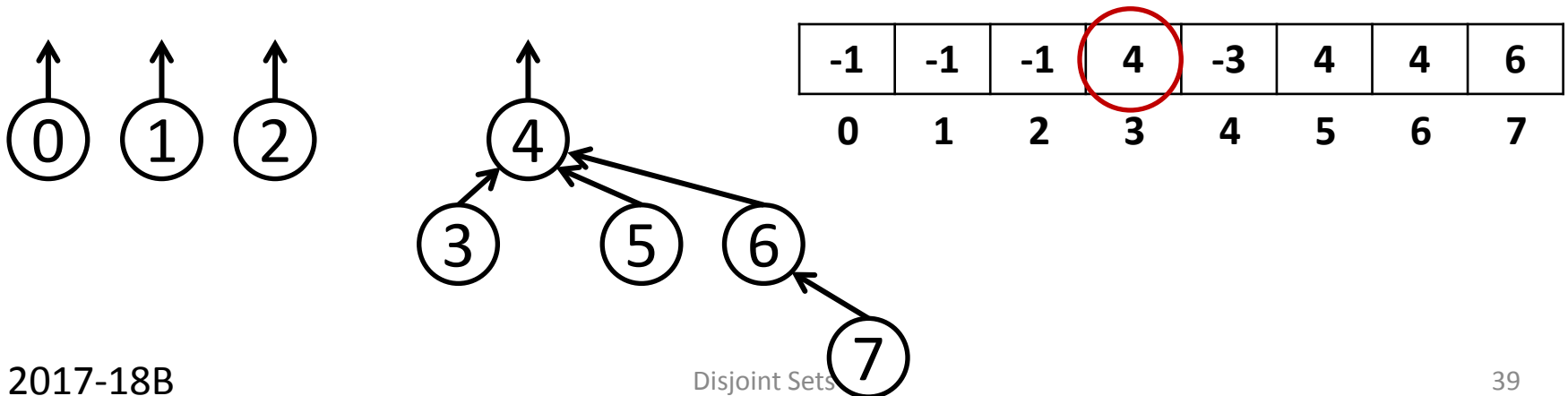


# Union-by-Height Example

- Before  $\text{Union}(3,4)$ :  $s[4] < s[3]$  (i.e. case 1)



- After  $\text{Union}(3,4)$ :  $s[3] = 4$



# Disjoint-Set Theorem

- **Theorem:** Union-by-height guarantees that all the trees have depth at most  $O(\log N)$ , where  $N$  is the total number of elements.
- **Lemma 1.** For any root  $x$ ,  $\text{size}(x) \geq 2^{\text{height}(x)}$ , where  $\text{size}(x)$  is the number of nodes of  $x$ 's tree, including  $x$ .



# Breaking News!!

---

## PHYSICS

### Mathematicians Just Discovered an 'Astonishing' New Way to Multiply Large Numbers

PETER DOCKRILL 9 APR 2019

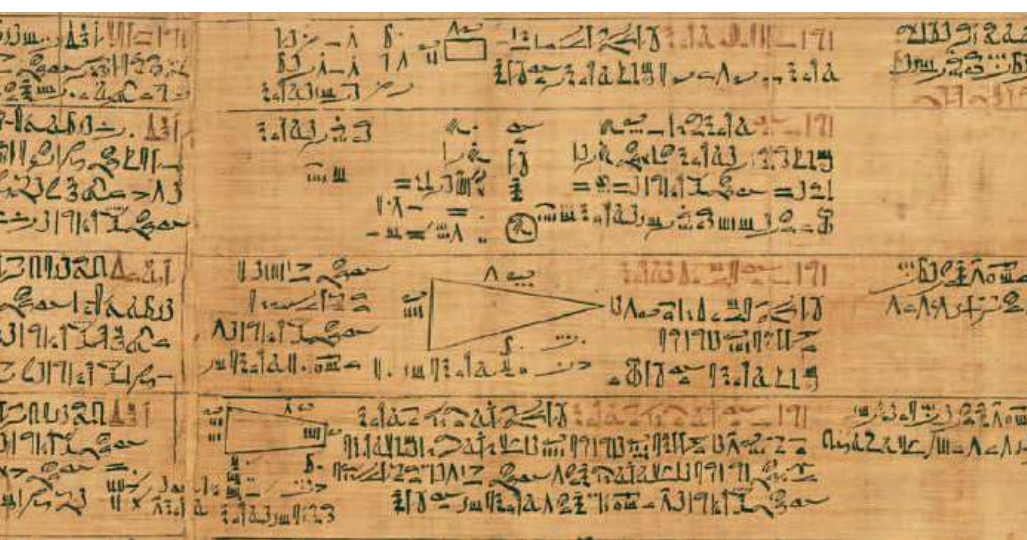
A pair of mathematicians from Australia and France have devised a new way to multiply numbers together, while solving an algorithmic puzzle that has perplexed some of the greatest math minds for almost half a century.

For most of us, the way we multiply relatively small numbers is by remembering our times tables – an incredibly handy aid first pioneered by the Babylonians some 4,000 years ago.

But what if the numbers get bigger? Well, if the figures get unwieldy – and assuming we don't have a calculator or computer, of course – most of us would then turn to long multiplication: another useful trick we learn in school, and a trusty technique for multiplying basically any two numbers together.

There's just one problem with long multiplication. It's slow.

<https://www.sciencealert.com/mathematicians-just-discovered-an-astonishing-new-way-to-multiply-numbers-together>



## Copper Plate Multiplication

What is going on here? Describe the 'method' to someone else in writing, so that they can use it on their own multiplications of different numbers of digits.

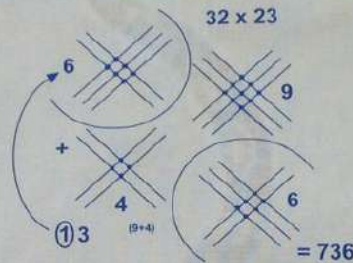
		7	9	6	4	5			
		6	4	7	8	9			
<hr/>									
			3	0					
			2	4	2	0			
			3	6	1	6	3	5	
		5	4	2	4	2	8	4	0
4	2	3	6	4	2	3	2	4	5
		2	8	6	3	4	8	3	6
			4	9	7	2	5	4	
				5	6	8	1		
				6	3				
<hr/>									
5	1	6	0	1	1	9	9	0	5



HOW DO YOU MULTIPLY  $32 \times 23 = ?$   $8 \times 8 = 64$ ; I ate and ate

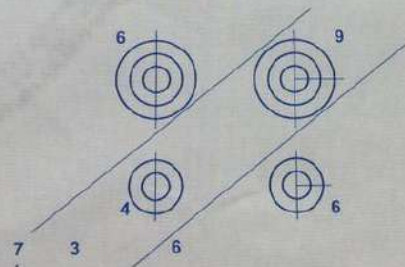
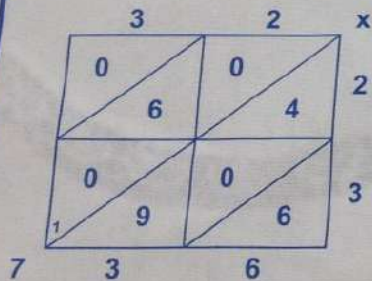
functional maths, numeracy

$$\begin{array}{r}
 32 \\
 \times 23 \\
 \hline
 640 \quad (32 \times 20) \\
 96 \quad (32 \times 3) \\
 \hline
 736
 \end{array}$$



	30	2	x
640	600	40	20
96	90	6	3
736			

You do the maths...

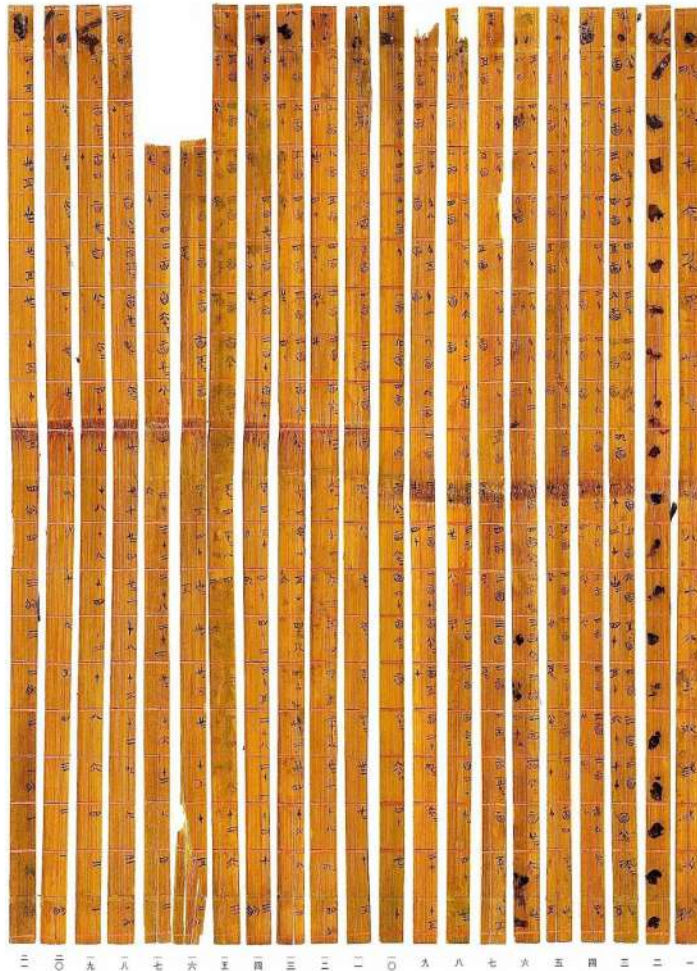


HALVE	DOUBLE
32	x 23
16	46
8	92
4	184
2	368
①	736
	736





# What is *the* fastest algorithm to multiply two $n$ -digit numbers?



MULTIPLICATION TABLE. 11

12. Repeat the

MULTIPLICATION TABLE.

2 times 1 are 2	3 times 1 are 3	4 times 1 are 4
2 times 2 are 4	3 times 2 are 6	4 times 2 are 8
2 times 3 are 6	3 times 3 are 9	4 times 3 are 12
2 times 4 are 8	3 times 4 are 12	4 times 4 are 16
2 times 5 are 10	3 times 5 are 15	4 times 5 are 20
2 times 6 are 12	3 times 6 are 18	4 times 6 are 24
2 times 7 are 14	3 times 7 are 21	4 times 7 are 28
2 times 8 are 16	3 times 8 are 24	4 times 8 are 32
2 times 9 are 18	3 times 9 are 27	4 times 9 are 36
2 times 10 are 20	3 times 10 are 30	4 times 10 are 40
2 times 11 are 22	3 times 11 are 33	4 times 11 are 44
2 times 12 are 24	3 times 12 are 36	4 times 12 are 48
5 times 1 are 5	6 times 1 are 6	7 times 1 are 7
5 times 2 are 10	6 times 2 are 12	7 times 2 are 14
5 times 3 are 15	6 times 3 are 18	7 times 3 are 21
5 times 4 are 20	6 times 4 are 24	7 times 4 are 28
5 times 5 are 25	6 times 5 are 30	7 times 5 are 35
5 times 6 are 30	6 times 6 are 36	7 times 6 are 42
5 times 7 are 35	6 times 7 are 42	7 times 7 are 49
5 times 8 are 40	6 times 8 are 48	7 times 8 are 56
5 times 9 are 45	6 times 9 are 54	7 times 9 are 63
5 times 10 are 50	6 times 10 are 60	7 times 10 are 70
5 times 11 are 55	6 times 11 are 66	7 times 11 are 77
5 times 12 are 60	6 times 12 are 72	7 times 12 are 84
8 times 1 are 8	9 times 1 are 9	10 times 1 are 10
8 times 2 are 16	9 times 2 are 18	10 times 2 are 20
8 times 3 are 24	9 times 3 are 27	10 times 3 are 30
8 times 4 are 32	9 times 4 are 36	10 times 4 are 40
8 times 5 are 40	9 times 5 are 45	10 times 5 are 50
8 times 6 are 48	9 times 6 are 54	10 times 6 are 60
8 times 7 are 56	9 times 7 are 63	10 times 7 are 70
8 times 8 are 64	9 times 8 are 72	10 times 8 are 80
8 times 9 are 72	9 times 9 are 81	10 times 9 are 90
8 times 10 are 80	9 times 10 are 90	10 times 10 are 100
8 times 11 are 88	9 times 11 are 99	10 times 11 are 110
8 times 12 are 96	9 times 12 are 108	10 times 12 are 120

This beautiful table appeared in Roswell W Smith's *Practical and Mental Arithmetic, on a New Plan, in Which Mental Arithmetic is Combined with the Use of a Slate...*, which was a soaringly successful book, this being the 53rd edition (!) and printed in Hartford in 1836. (Smith was a busy man, writing several other standard and widely distributed works in geography and grammar).

# Learning at Scale

11

A STUDY IN MACHINE-AIDED LEARNING

by

Chung Laung Liu

Submitted to the Department of Electrical Engineering  
on May 21, 1960 in partial fulfillment of the requirements  
for the degree of Master of Science.

## ABSTRACT

The conventional teaching machines are studied. The advantages and the possibility of using a computer as a teaching machine are investigated. A program using an IBM 704 computer to teach arithmetic and a program using an IBM 704 computer to teach matrix algebra are suggested.

Thesis Supervisor: Ronald A. Howard

Title: Assistant Professor of  
Electrical Engineering

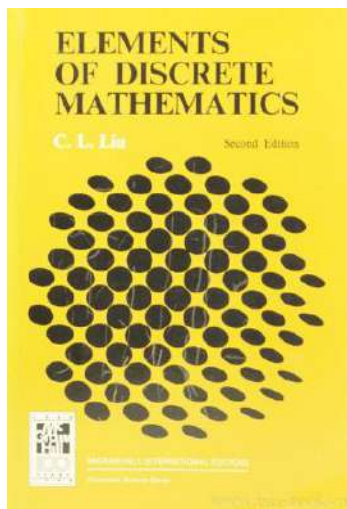


**Chung Laung (Dave) Liu**

劉炯朗, who once led NTHU and the CS Dept at UIUC, wrote one of the first and very influential books on discrete mathematics and computer science. His Master Thesis at MIT was the first work on computer-based learning.

[https://en.wikipedia.org/wiki/Chung\\_Laung\\_Liu](https://en.wikipedia.org/wiki/Chung_Laung_Liu)

[www.cityu.edu.hk/lib/about/event/ls\\_liu/ls\\_liu.pps](http://www.cityu.edu.hk/lib/about/event/ls_liu/ls_liu.pps)



# Back to Square One



## Exercise 8.11

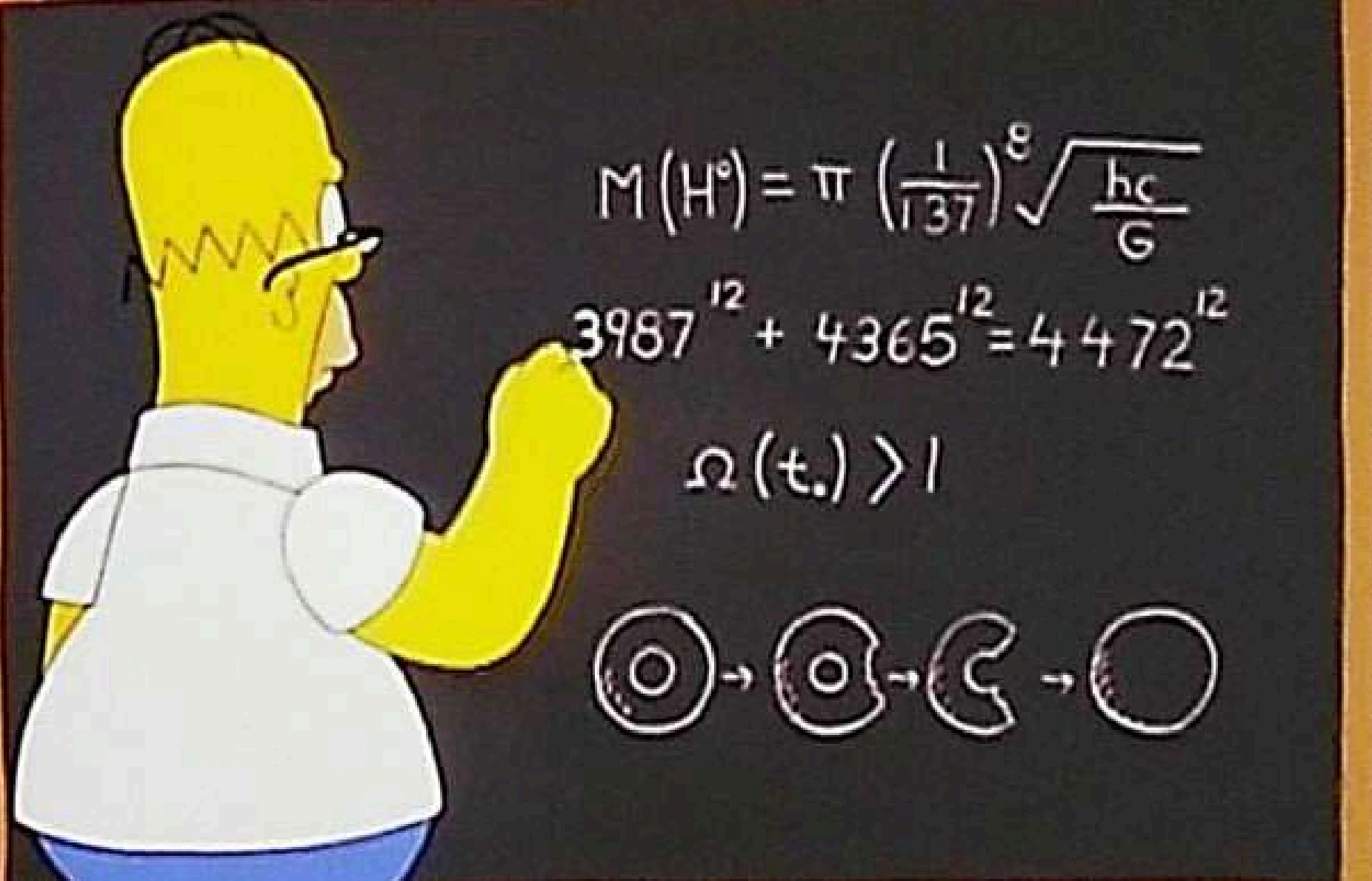
We study the problem of computing  $x^n$  for given  $x$  and  $n$ , assuming that all intermediate results of computations are available to us.

For a given integer  $n$ , let  $b_k b_{k-1} \dots b_1 b_0$  be its binary number representation. For each  $b_i$ , we will replace  $b_i$  by  $SX$  if  $b_i = 1$  and replace  $b_i$  by  $S$  if  $b_i = 0$ . For example, since the binary number representation for 29 is 11101, we obtain the sequence  $SXSXSXSSX$ . Removing the leading  $SX$ , we obtain the sequence  $SXSXSSX$ . Substitute *each S by square*, and *each X by multiply by x*, where square means to compute the square of the current result, and multiply by  $x$  means to multiply the current result by  $x$ . Starting with  $x$  as the current result, prove that the algorithm is indeed correct.

Given an integer  $n$ , an *addition chain* for  $n$  is a sequence of integers  $a_0 a_1 \dots a_m$  such that  $a_0 = 1$ ,  $a_m = n$  and  $a_i = a_j + a_k$  for  $k \leq j < i$ .

What is the relationship between *addition chains* and evaluation of powers of  $x$ ?





<http://www.npr.org/sections/krulwich/2014/05/08/310818693/did-homer-simpson-actually-solve-fermat-s-last-theorem-take-a-look>

<http://www.math.utoronto.ca/barbeau/fn41.pdf>



## Fermat's Last Theorem:

$x^n + y^n = z^n$  has no integer solution for  $n > 2$

<http://www.npr.org/sections/krulwich/2014/05/08/310818693/did-homer-simpson-actually-solve-fermat-s-last-theorem-take-a-look>

<http://www.math.utoronto.ca/barbeau/fn41.pdf>





**Fermat's Library**

@fermatlibrary

Follow



## Here's a peculiar Sorting Algorithm

For every element on the sequence the program does this

1. Sleeps for  $X$  seconds
2. Prints  $X$

The clock starts simultaneously for all elements

5:38 AM - 16 Mar 2019

**390** Retweets **1,943** Likes



# Learning at Scale



Good news! Contact me for paid work on Nemo Bot to promote Computer Science

