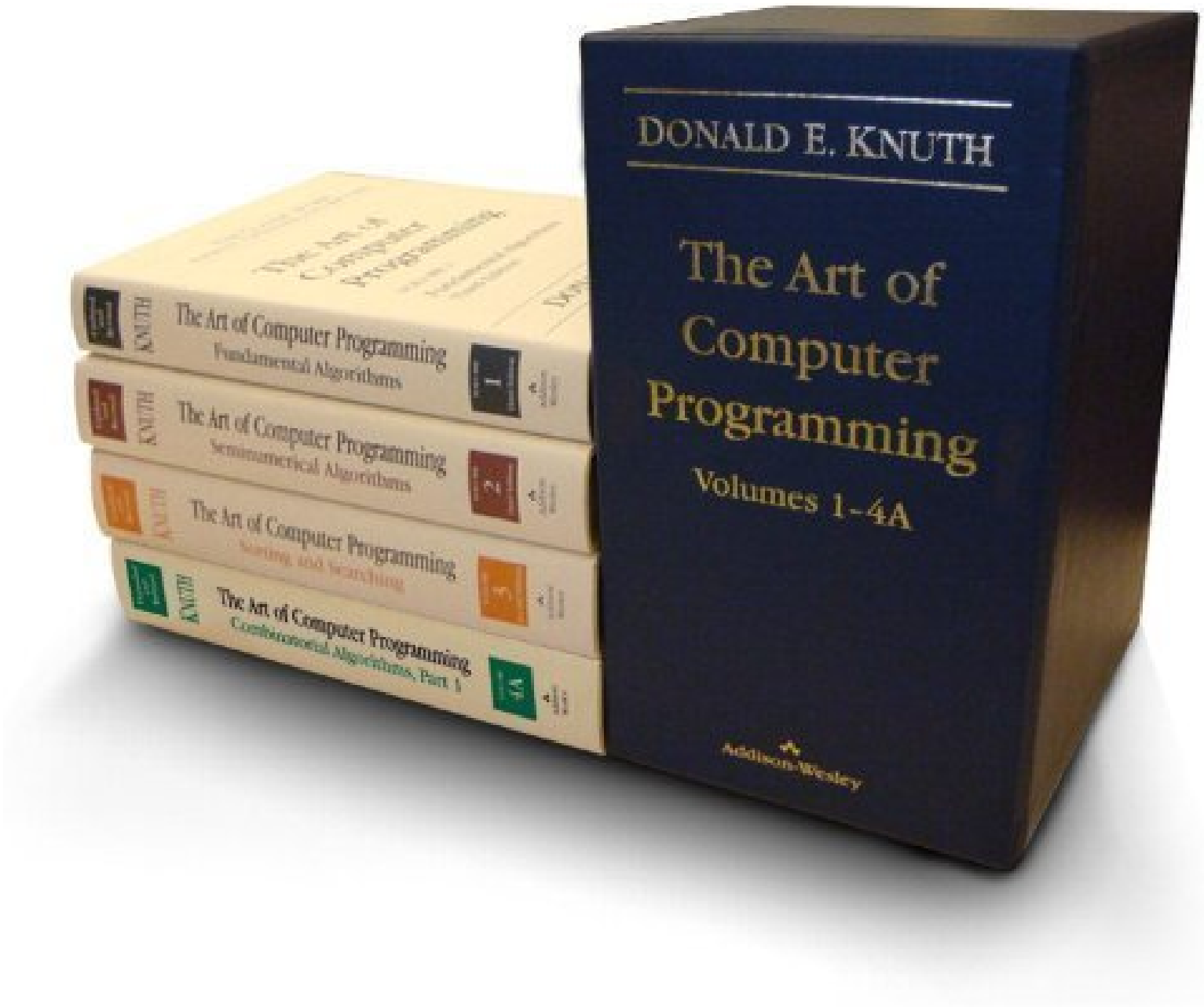
ACM Turing Award Winner 1974: Donald Ervin Knuth

Venkatesh Raman
The Institute of Mathematical Sciences, Chennai

Talk at ACM Chennai Chapter
September 21, 2017

The Turing Award Citation 1974

..... for a number of major contributions to analysis of algorithms and the design of programming languages, and in particular for his most signification contributions to the "art of computer programming" through his series of well-known books. The collection of techniques, algorithms, and relevant theory in these books have served as a focal point for developing curricula and as an organizing influence on computer science.


DONALD E. KNUTH
The Art of Computer Programming
Volumes 1-4A
Addison-Wesley
The Art of Computer Programming
Fundamental Algorithms
The Art of Computer Programming
Seminumerical Algorithms
The Art of Computer Programming
Sorting and Searching
The Art of Computer Programming
Combinatorial Algorithms, Part 1

A man who has straddled

A man who has straddled

- Theory and Practice

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy
  -fundamental research vs books; his research work opened several subareas in computer science, but retired early in 1993 to work on his AOCP volumes

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy
  -fundamental research vs books; his research work opened several subareas in computer science, but retired early in 1993 to work on his AOCP volumes
- Art and Science

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy
  -fundamental research vs books; his research work opened several subareas in computer science, but retired early in 1993 to work on his AOCP volumes
- Art and Science
  -Art of computer programming, Languages, History, Music

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy
  -fundamental research vs books; his research work opened several subareas in computer science, but retired early in 1993 to work on his AOCP volumes
- Art and Science
  -Art of computer programming, Languages, History, Music
- Science and Religion

A man who has straddled

- Theory and Practice
  -Analysis of Algorithms vs TEX; writes many lines of code every day
- Research and Pedagogy
  -fundamental research vs books; his research work opened several subareas in computer science, but retired early in 1993 to work on his AOCP volumes
- Art and Science
  -Art of computer programming, Languages, History, Music
- Science and Religion
  - is part of bible study every Sunday, written a book on Chapter 3, Verse 16 on bible books

Childhood and Education

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for “discrete mathematics”

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for "discrete mathematics"
- Developed a hobby of writing programs for IBM 650 in the computer centre

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for "discrete mathematics"
- Developed a hobby of writing programs for IBM 650 in the computer centre
- Learnt the source code of 'internal translator' for IBM 650

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for "discrete mathematics"
- Developed a hobby of writing programs for IBM 650 in the computer centre
- Learnt the source code of 'internal translator' for IBM 650
- Developed fascination for sources of codes, papers, terminologies

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for "discrete mathematics"
- Developed a hobby of writing programs for IBM 650 in the computer centre
- Learnt the source code of 'internal translator' for IBM 650
- Developed fascination for sources of codes, papers, terminologies
- Case Institute decided to award both BS and MS in 1960

Childhood and Education

Born in 1938 in Milwaukee, Wisconsin, USA. Went to a lutheran church school till 12th.
Was interested in music and physics, but completed undergrad in Math in Case Institute of Technology, Ohio

- Developed a liking for "discrete mathematics"
- Developed a hobby of writing programs for IBM 650 in the computer centre
- Learnt the source code of 'internal translator' for IBM 650
- Developed fascination for sources of codes, papers, terminologies
- Case Institute decided to award both BS and MS in 1960
- Wrote ALGOL compilers to Burroughs corporation

PhD years

California Institute of Technology (Caltech)

California Institute of Technology (Caltech)

- Started on some block design problem with Marshall Hall Jr.

PhD years

California Institute of Technology (Caltech)

- Started on some block design problem with Marshall Hall Jr.
- Digressed, found and solved a different problem (on projective planes) for his PhD

PhD years

California Institute of Technology (Caltech)

- Started on some block design problem with Marshall Hall Jr.
- Digressed, found and solved a different problem (on projective planes) for his PhD in an hour.

PhD years

California Institute of Technology (Caltech)

- Started on some block design problem with Marshall Hall Jr.
- Digressed, found and solved a different problem (on projective planes) for his PhD in an hour.
- Continued to write software

PhD years

California Institute of Technology (Caltech)

- Started on some block design problem with Marshall Hall Jr.
- Digressed, found and solved a different problem (on projective planes) for his PhD in an hour.
- Continued to write software
- Obtained PhD in 1963

After PhD at Caltech

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section
- Beginning of the Compiler book project with Addison-Weseley which was later envisaged into a 7 volume book.

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section
- Beginning of the Compiler book project with Addison-Weseley which was later envisaged into a 7 volume book.
- 1967 peak year for him!

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section
- Beginning of the Compiler book project with Addison-Weseley which was later envisaged into a 7 volume book.
- 1967 peak year for him!
- Resigned from JACM, CACM editorial boards

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section
- Beginning of the Compiler book project with Addison-Weseley which was later envisaged into a 7 volume book.
- 1967 peak year for him!
- Resigned from JACM, CACM editorial boards (but gradually got into many more later!),

After PhD at Caltech

- Remained in math faculty; continued to publish papers and write software
- Consultant for Burroughs Corporation
- Editor for ACM publications (JACM 1964-1967, CACM 1966) working on their programming languages section
- Beginning of the Compiler book project with Addison-Weseley which was later envisaged into a 7 volume book.
- 1967 peak year for him!
- Resigned from JACM, CACM editorial boards (but gradually got into many more later!),
- Left Caltech for a brief stint at the Institute for Defense Analyses' Communications Research Division (NSA/Crypto).

Stanford

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)

Stanford

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.

Stanford

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974

Stanford

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics
  - Became the first CS endowed chair in 1977

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics
  - Became the first CS endowed chair in 1977
  - Supervised around 30 PhD students.

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics
  - Became the first CS endowed chair in 1977
  - Supervised around 30 PhD students.
    Pratt, Fredman, Sedgewick, Vitter are some of his students.

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics
  - Became the first CS endowed chair in 1977
  - Supervised around 30 PhD students.
    Pratt, Fredman, Sedgewick, Vitter are some of his students.
  - Was in the editorial board for more than 30 journals.

- Moved to Stanford in 1968 (Bob Floyd also moved around the same time)
  - Released TAOCP Volumes 1-3 during 1968, 69 and 73.
  - Got Turing Award in 1974
  - introduced courses on data structures, concrete mathematics
  - Became the first CS endowed chair in 1977
  - Supervised around 30 PhD students.
    Pratt, Fredman, Sedgewick, Vitter are some of his students.
  - Was in the editorial board for more than 30 journals.
  - Became professor emeritus of The Art of Computer Programming (TAOCP) in 1993

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science
- Evolved into a project consisting on 7 volume series on *The Art of Computer Programming*
- Volumes 1-3 published in 1968, 1969 and 1973

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science
- Evolved into a project consisting on 7 volume series on *The Art of Computer Programming*
- Volumes 1-3 published in 1968, 1969 and 1973
  Versions of Volume 4 started appearing from 2005 as fascicles.

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science
- Evolved into a project consisting on 7 volume series on *The Art of Computer Programming*
- Volumes 1-3 published in 1968, 1969 and 1973
  Versions of Volume 4 started appearing from 2005 as fascicles.
- Took 10 years off to develop digital typography: TEX(for document preparation) and METAFONT (for alphabet design)

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science
- Evolved into a project consisting on 7 volume series on *The Art of Computer Programming*
- Volumes 1-3 published in 1968, 1969 and 1973
  Versions of Volume 4 started appearing from 2005 as fascicles.
- Took 10 years off to develop digital typography: TEX(for document preparation) and METAFONT (for alphabet design)
- Byproducts: WEB and CWEB languages for structured programming and accompanying methodology for Literate Programming.

TAOCP and TEX

- Began a project in 1962 on compilers and programming techniques with Addison-Wesley
- When he started writing about Sorting, he realized that very little correct literature was available.
- So he decided to take a qualitative approach and wanted to organize things and tell the real story of computer science
- Evolved into a project consisting on 7 volume series on *The Art of Computer Programming*
- Volumes 1-3 published in 1968, 1969 and 1973
  Versions of Volume 4 started appearing from 2005 as fascicles.
- Took 10 years off to develop digital typography: TEX(for document preparation) and METAFONT (for alphabet design)
- Byproducts: WEB and CWEB languages for structured programming and accompanying methodology for Literate Programming. (He believed that programs are ALSO written for humans to read/understand/debug.)

The TAOCP

- Volume 1 – Fundamental Algorithms, published in 1968, 1969, second edition in 1973, 1974 and third in 1997
- Volume 2 – Seminumerical Algorithms, published in 1969, 1971, second edition in 1981 and third in 1997
- Volume 3 – Sorting and Searching, published in 1973, 1975, second edition in 1998.
- Volume 4A – Combinatorial Algorithms, Part I, 2011.
- Volume 4, Fascicle 6 – Satisfiability, 2015

TAOCP continued

TAOCP continued

- More than one million copies printed; Translated into 9 languages

TAOCP continued

- More than one million copies printed; Translated into 9 languages
- He was paying $2.56 for any first error spotted in the book(s).

TAOCP continued

- More than one million copies printed; Translated into 9 languages
- He was paying $2.56 for any first error spotted in the book(s).
- Exercises and Gradings for the problems became important component of the books
- Open problems from the first edition of Volume 3 triggered several PhD theses.
- "Go to" place for most of the material in these volumes.
- In the second edition of Volume 3, in the index, had the names of every author in their native language.

Prediction, *see* Forecasting.
Preferential arrangements, 194.
Prefetching, 369–373.
Prefix, 492.
Prefix code, 452–453.
  for all nonnegative integers, 6.
Prefix search, *see* Trie search.
Preorder merge, 307–309.
Prestet, Jean, 24.
Prime numbers, 156, 516, 529, 557, 627.
Primitive comparator networks, 240, 668.
Principle of optimality, 363, 438.
Pring, Edward John, 564.
Prins, Jan Fokko, 618.
Priority deques, 157.
Priority queues, 148–152, 156–158, 253, 646, 705.
  merging, 150, 157.
Priority search trees, 578.
Probability density functions, 177.
Probability distributions, 105, 399–401.
  beta, 586.
  binomial, 100–101, 341, 539, 565.
  fractal, 400.
  normal, 45, 69, 650.
  Pareto, 401, 405, 710.
  Poisson, 555.
  random, 458.
  uniform, 6, 16, 20, 47, 127, 606.
  Yule, 401, 405.
  Zipf, 400, 402, 435, 455.
Probability generating functions, 15–16, 102, 104, 135, 177, 425, 490, 539, 553, 555, 739.
Prodinger, Helmut, 576, 634, 644, 648, 726, 726.
Product of consecutive binomial coefficients, 612.
Proof of algorithms, 49–51, 112–113, 315, 323, 355, 677.
Prusker, Francis, 377.
Prywes, Noah Shmarya, 578.
Pseudolines, 670.
Psi function $\psi(z)$, 637, 751.
Puech, Claude Henri Clair Marie Jules, 565, 566, 576.
Pugh, William Worthington, Jr., 213, 478.
Punched cards, 169–170, 175, 383–385.

q-multinomial coefficients, 32.
q-nomial coefficients, 32, 594, 595.
q-series, 20, 32, 594–596, 644.
Quadrangle inequality, 457.
Quadratic probing, 551.
Quadratic selection, 141.
Quadruple systems, 581, 746.
Quadtrees, 565–566, 581, 745–746.
Queries, 559–582.
Questionnaires, 183.
Queues, 135, 148–149, 156, 171, 299, 310, 322–323.
Quickfind, 136.
  median-of-three, 634.
Quicksort, 113–122, 135–138, 148, 159, 246, 349–351, 356, 381, 382, 389, 389, 431, 698.
  binary, *see* Radix exchange.
  median-of-three, 122, 136, 138, 381, 382.
  multikey, 389, 633, 728.
  with equal keys, 136, 635–636.

Rabbits, 424.
Rabin, Michael Oser (מיכאל עזר רבין), 242.
Radix-2 sorting, 387.
Radix exchange sort, 122–128, 130–133, 136–138, 159, 177, 351, 382, 389, 500–501, 509, 698.
  with equal keys, 127–128, 137.
Radix insertion sort, 176–177.
Radix list sort, 171–175, 382.
Radix sorting, 5, 169–179, 180–181, 343–348, 351, 359, 374, 381, 385, 389, 421, 502, 698.
  dual to merge sorting, 345–348, 359.
Radke, Charles Edwin, 297.
Räihä, Kari-Jouko, 717.
Railway switching, 168.
Rains, Eric Michael, 611.
Rais, Bonita Marie, 726.
Raman Rajeev, 634.
Raman, Venkatesh (வெங்கடேஷ் ராமன்), 655.
Ramanan, Prakash Viriyur (விரியூர் வரதராஜ ராமனன்), 218.
Ramanujan Iyengar, Srinivasa (ஸ்ரீனிவாஸ ராமானுஜன் ஐயங்கார்), function $Q(n)$, 701.
Ramshaw, Lyle Harold, 729.
Random data for sorting, 20, 47, 76, 383, 391.
Random probability distribution, 458.
Random probing, independent, 548, 555.
  with secondary clustering, 548, 554.
Randomized adversary: An adversary that flips coins, 219.
Randomized algorithms, 121–122, 351, 455, 517, 519, 557–558.
Randomized binary search trees, 478.
Randomized data structures, 478.
Randomized striping, 371–373, 379, 698.
Randrianarimanana, Bruno, 713.
Raney, George Neal, 297, 298.
Range queries, 559, 578.
RANK field, 471, 476, 479, 713, 718.
Ranking, 181, *see* Sorting.
Raver, Norman, 729.
Ravikumar, Balasubramanian (பாலசுப்ரமணியன் ரவிகுமார்), 673.
Rawlings, Don Paul, 595.

César, Yves, 193, 279.
Chaining, 520–525, 542–544, 547, 553, 557.
  to reduce seek time, 368–369.
Chakravarti, Gurugovinda (গুরুগোবিন্দ চক্রবর্ত্তী), 23.
Chandra, Ashok Kumar (अशोक कुमार चन्द्रा), 422.
Chang, Shi-Kuo (張系國), 458.
Chartres, Bruce Aylwin, 156.
Chase, Stephen Martin, 196.
Chazelle, Bernard Marie, 583.
Chebyshev, Pafnutii Lvovich (Чебышев, Пафнутий Львович), 395.
  polynomials, 296, 685.
Chen, Wen-Chin (陳文進), 548.
Cherkassky, Boris Vasilievich (Черкасский, Борис Васильевич), 152.
Chessboard, 14, 46–47, 69.
Choice of data structure, 95–96, 141, 151–152, 163–164, 170–171, 459, 561–567.
Chow, David Kuo-kien, 578.
Christen, Claude André, 204, 658.
Chronological order, 372, 379.
Chung, Fan Rong King (鍾金芳蓉), 402.
Chung, Moon Jung (정문정 = 鄭文楨), 673.
Church, Randolph, 669.
CI: MIX's comparison indicator, 6.
Cichelli, Richard James, 513.
Circular lists, 407, 729.
Clausen, Thomas, 157.
Cleave, John Percival, 400.
Clément, Julien Stephane, 728.
Cliques, 9.
Closest match, search for, 9, 394, 408, 563, 566, 581.
CMath: Concrete Mathematics, a book by R. L. Graham, D. E. Knuth, and O. Patashnik.
Comp. J.: The Computer Journal, a publication of the British Computer Society since 1958.
Comparator modules, 221, 234, 241.
Comparison counting sort, 75–80, 382, 387.
Comparison-exchange tree, 196.
Comparison matrix, 188.
Comparison of algorithms, 151, 324–338, 347–348, 380–383, 471, 545–547.
Comparison of keys, 4.
  minimizing, 180–247, 413, 425, 549.
  multiprecision, 6, 136, 169.
  parallel, 113, 222–223, 228–229, 235, 390, 425, 671.
  searching by, 398–399, 409–491, 546–547.
  sorting by, 80–122, 134–168, 180–197, 219–343, 348–383.
Comparison trees, 181–182, 192–197, 217, 219–220, 411–417.
Compiler techniques, 2–3, 426, 532.
Complement notations, 177.
Complementary pairs, 9.
Complemented block designs, 581.
Complete binary trees, 144, 152–153, 158, 211, 217, 253–254, 258, 267, 425.
Complete P-ary tree, 361, 697.
Complete ternary trees, 157.
Complex partitions, 21.
Complexity analysis of algorithms, 168, 178–179, 180–247, 302–311, 353–356, 374–378, 388, 412–413, 425, 491, 539–541, 549, 578.
Components of graphs, 189.
Compound attributes, 564, 566–567.
Compound leaf of a tree, 688.
Compressed tries, 507.
  dynamic, 722.
Compression of data, 453, 512.
Compromise merge, 297.

Received: from CS.Stanford.EDU (CS.Stanford.EDU [171.64.64.64])
        by Sunburn.Stanford.EDU (8.8.7/8.8.8) with ESMTP id WAA25897
        for <winkler@sunburn.Stanford.EDU>; Mon, 23 Feb 1998 22:54:08 -0800 (PST)
Received: from imsc3.imsc.ernet.in (imsc3.imsc.ernet.in [202.41.95.7])
        by CS.Stanford.EDU (8.8.8/8.8.8) with ESMTP id WAA08884
        for <taocp@cs.stanford.edu>; Mon, 23 Feb 1998 22:55:24 -0800 (PST)
Received: (from vraman@localhost) by imsc3.imsc.ernet.in (8.7.5/8.7.3) id MAA23860 for taocp@cs.stanford.edu; Tue, 24 Feb 1998 12:24:00 +0530 (GMT+05:30)
Date: Tue, 24 Feb 1998 12:24:00 +0530 (GMT+05:30)
From: Venkatesh Raman <vraman@imsc.ernet.in>
Message-Id: <199802240654.MAA23860@imsc3.imsc.ernet.in>
To: taocp@CS.Stanford.EDU
Subject: A possible unreported error

Dear Prof Knuth,

I haven't seen the following (possible) error
in Volume 3, reported in your home page. Maybe
it is not an error or already someone has reported
which I couldn't locate.

In section 5.2.3, exercise 35 (Page 159) gives a procedure for
deleting an element (node P) from a leftist heap:

        replace P by merger of left(P) and right(P) and fix the
        distance field of its ancestors.

I don't think it is enough. The tree obtained after replacing
P by merger of left(P) and right(P) need not even be a leftist
tree (not hard to constuct such examples). So, one has to check
the "right distance bigger" condition also and swap "left-right"
subtrees if necessary in the path where we change the distance field,
as you'd do for merge operation. One could do everything in O(log n)
time, of course.

Venkatesh

Thank goodness you reported this on the very day
I was finishing the new Vol 3 ... 24 hours later
would probably have been too late!

I agree that the exercise is badly worded,
although the answer does have this in mind
in the case $d_1 > d_0$.

I have J of Alg and BIT, TCS and Algorithmica

Thank goodness you reported this on the very day
I was finishing the new Vol 3 ... 24 hours later
would probably have been too late!

I agree that the exercise is badly worded,
although the answer does have this in mind
(in the case $d_1 > d_0$)

About your papers: I have J of Alg and BIT, TCS and Algorithmica
but would appreciate reprints from
because they will help me decide what to put in the
3rd edition I had little time for major changes and I cited
edition about 15 years from now. In the 2nd
only your WADS 91 paper where it refers to
heapsort with equal keys. I am now on my way
to the library to see if the WADS 91 paper has a
journal version, in which case I will change the
citation from LNCS to the appropriate j

Cordially don knuth

Research Highlights

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).
- Involution – a permutation π such that $\pi = \pi^{-1}$. (Example: 21345)

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).
- Involution – a permutation π such that $\pi = \pi^{-1}$. (Example: 21345)
- Young's tableaux of shape $(n_1, n_2, \ldots n_k)$ where $n_1 \geq n_2 \geq \ldots n_k$ (ON the BOARD)

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).
- Involution – a permutation π such that $\pi = \pi^{-1}$. (Example: 21345)
- Young's tableaux of shape $(n_1, n_2, \ldots n_k)$ where $n_1 \geq n_2 \geq \ldots n_k$ (ON the BOARD)
- Robinson-Schensted – Number of involutions on n elements = Number of tableaux on n cells.

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).
- Involution – a permutation π such that $\pi = \pi^{-1}$. (Example: 21345)
- Young's tableaux of shape $(n_1, n_2, \ldots n_k)$ where $n_1 \geq n_2 \geq \ldots n_k$ (ON the BOARD)
- Robinson-Schensted – Number of involutions on n elements = Number of tableaux on n cells. (More generally, each permutation corresponds to a pair of tableau of the same shape, and if $\pi = (P, Q)$, then $\pi^{-1} = (Q, P)$.)

Robinson-Schensted-Knuth or RSK correspondence between the set of involutions on n elements and the set of young tableaux on n cells.

- Let $S = \{1, 2, \ldots n\}$.
- Permutation – a bijection from S to S (e.g. 34152 is a permutation on 5 elements where 1 is mapped to 3, 2 to 4 etc).
- Involution – a permutation π such that $\pi = \pi^{-1}$. (Example: 21345)
- Young's tableaux of shape $(n_1, n_2, \ldots n_k)$ where $n_1 \geq n_2 \geq \ldots n_k$ (ON the BOARD)
- Robinson-Schensted – Number of involutions on n elements = Number of tableaux on n cells. (More generally, each permutation corresponds to a pair of tableau of the same shape, and if $\pi = (P, Q)$, then $\pi^{-1} = (Q, P)$.)
- Knuth gave an algorithmic proof and extended this for permutations on multisets in a 1971 paper.

'This purely combinatorial algorithm, and this correspondence has had profound repurcussions in algebraic geometry, the theory of symmetric functions, invariant theory and representation theory. In fact, it plays the role of a unifying principle in representation theory of symmetric groups, polynomial representations of matrix groups and symmetric function theory.'

'This purely combinatorial algorithm, and this correspondence has had profound repurcussions in algebraic geometry, the theory of symmetric functions, invariant theory and representation theory. In fact, it plays the role of a unifying principle in representation theory of symmetric groups, polynomial representations of matrix groups and symmetric function theory.' Amritanshu Prasad (IMSc)

Linear Probing (Hashing)

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?
- For example $\delta(m, 1) =$

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?
- For example $\delta(m, 1) = 1$

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?
- For example $\delta(m, 1) = 1$ and $\delta(m, m) =$

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?
- For example $\delta(m, 1) = 1$ and $\delta(m, m) = (1 + 2 + 3 + \ldots m - 1)/m = (m - 1)/2$.

Linear Probing (Hashing)

Searching – Given a set of keys, organize the set so that given an element x check whether x is in the set can be answered fast.

- Linear search, binary search
- Hashing: $h : U \to T$, $|T| = m$.
- Each item x is stored in $h(x)$ if that position is non-empty and otherwise, follow in a linear (circular) fashion to store it in the next available empty spot (we will assume that the set we store is of size at most m so at least one location is empty).
- Question: If $x_1 x_2, \ldots x_m$ is a sequence of inserts, where each $x_i \in T$ is a random element (and $h(x) = x$), what is the expected number $\delta(m, n)$ of steps for the n-th insert/search?
- For example $\delta(m, 1) = 1$ and $\delta(m, m) = (1 + 2 + 3 + \ldots m - 1)/m = (m - 1)/2$.
- Computing $\delta(m, n)$ for intermediate values of n is quite non-trivial.

Linear probing analysis – Continued

- For example $\delta(m, 1) = 1$ and $\delta(m, m) = (1 + 2 + 3 + \ldots m - 1)/m = (m - 1)/2$.
- Knuth showed that

$$\begin{aligned} \delta(m, n) &= 1 + (n - 1)/2m + (n - 1)(n - 2)/2m^2 \\ &+ (n - 1)(n - 2)(n - 3)/2m^3 + \ldots \end{aligned}$$

Linear probing analysis – Continued

- For example $\delta(m,1) = 1$ and $\delta(m,m) = (1+2+3+\ldots m-1)/m = (m-1)/2$.
- Knuth showed that

$$\begin{aligned}\delta(m,n) &= 1 + (n-1)/2m + (n-1)(n-2)/2m^2 \\ &+ (n-1)(n-2)(n-3)/2m^3 + \ldots\end{aligned}$$

- Encountered a sum like the following

$$\sum_{r \geq 0} r \binom{n-1}{r} (r+1)^r (m-r-1)^n - r - 2$$

Linear probing analysis – Continued

- For example $\delta(m, 1) = 1$ and $\delta(m, m) = (1 + 2 + 3 + \ldots m - 1)/m = (m - 1)/2$.
- Knuth showed that

$$\begin{aligned} \delta(m, n) &= 1 + (n-1)/2m + (n-1)(n-2)/2m^2 \\ &+ (n-1)(n-2)(n-3)/2m^3 + \ldots \end{aligned}$$

- Encountered a sum like the following

$$\sum_{r \geq 0} r \binom{n-1}{r} (r+1)^r (m-r-1)^n - r - 2$$

which he managed to find a close form to derive the value for $\delta(m, n)$.

Linear probing analysis – Continued

- For example $\delta(m,1) = 1$ and $\delta(m,m) = (1+2+3+\ldots m-1)/m = (m-1)/2$.
- Knuth showed that

$$\begin{aligned}\delta(m,n) &= 1 + (n-1)/2m + (n-1)(n-2)/2m^2 \\ &+ (n-1)(n-2)(n-3)/2m^3 + \ldots\end{aligned}$$

- Encountered a sum like the following

$$\sum_{r\geq 0} r\binom{n-1}{r}(r+1)^r(m-r-1)^n - r - 2$$

  which he managed to find a close form to derive the value for $\delta(m,n)$.
- Gave him the idea of "Concrete Mathematics" (a course at Stanford, and he later wrote a book on the subject) where such finite sums are dealt with.

in Problem 1. Replacing r by $-k-2$ gives the desired expansion,

$$S_m = \sum_{k\geq 0}\binom{n+k}{k}\binom{n}{k}\frac{(-1)^k}{k+1}\sum_{j\geq 0}\binom{m}{j}\binom{-k-2}{j}^{-1}.$$

Now the $(k+1)^{-1}$ can be absorbed into $\binom{n}{k}$, as planned. In fact, it could also be absorbed into $\binom{-k-2}{j}^{-1}$. Double absorption suggests that even more cancellation might be possible behind the scenes. Yes — expanding everything in our new summand into factorials and going back to binomial coefficients gives a formula that we can sum on k:

They expect us to check this on a sheet of scratch paper.

$$\begin{aligned} S_m &= \frac{m!\,n!}{(m+n+1)!}\sum_{j\geq 0}(-1)^j\binom{m+n+1}{n+1+j}\sum_k\binom{n+1+j}{k+j+1}\binom{-n-1}{k} \\ &= \frac{m!\,n!}{(m+n+1)!}\sum_{j\geq 0}(-1)^j\binom{m+n+1}{n+1+j}\binom{j}{n}. \end{aligned}$$

The sum over all integers j is zero, by (5.24). Hence $-S_m$ is the sum for $j<0$.

To evaluate $-S_m$ for $j<0$, let's replace j by $-k-1$ and sum for $k\geq 0$:

$$\begin{aligned} S_m &= \frac{m!\,n!}{(m+n+1)!}\sum_{k\geq 0}(-1)^k\binom{m+n+1}{n-k}\binom{-k-1}{n} \\ &= \frac{m!\,n!}{(m+n+1)!}\sum_{k\leq n}(-1)^{n-k}\binom{m+n+1}{k}\binom{k-n-1}{n} \\ &= \frac{m!\,n!}{(m+n+1)!}\sum_{k\leq n}(-1)^k\binom{m+n+1}{k}\binom{2n-k}{n} \\ &= \frac{m!\,n!}{(m+n+1)!}\sum_{k\leq 2n}(-1)^k\binom{m+n+1}{k}\binom{2n-k}{n}. \end{aligned}$$

Finally (5.25) applies, and we have our answer:

KMP $O(n + m)$ Algorithm (string matching)

KMP $O(n+m)$ Algorithm (string matching)

Given two strings x and y of length m and n each over an alphabet Σ, determine whether the string x is a substring of y.

- Naive approach: $O(mn)$

KMP $O(n+m)$ Algorithm (string matching)

Given two strings x and y of length m and n each over an alphabet Σ, determine whether the string x is a substring of y.

- Naive approach: $O(mn)$
- Build an automata for accepting strings containing x, and then run y on the automata and see if it is accepted.

KMP $O(n+m)$ Algorithm (string matching)

Given two strings x and y of length m and n each over an alphabet Σ, determine whether the string x is a substring of y.

- Naive approach: $O(mn)$
- Build an automata for accepting strings containing x, and then run y on the automata and see if it is accepted.
- Building the automata takes $O(m^3|\Sigma|)$. Knuth, Morris and Pratt came up with a way to compute it in $O(m)$ time. (Running y on the automata takes $O(n)$ time.)

Other highlights

Other highlights

- Knuth-Bendix algorithm (Axiomatic Reasoning, term-rewriting)

Other highlights

- Knuth-Bendix algorithm (Axiomatic Reasoning, term-rewriting)
- $LR(k)$ Grammars (languages accepted by deterministic push down automaton). Practical applications for compiler constructions.

Books

- The Art of Computer Programming Volume 1-3, 4A, 4 fasciciles
- Concrete Mathematics (with Graham and Patashnik)
- Marriages Stables
- Axioms and Hulls

- Surreal Numbers
- Mathematical Analysis of Algorithms
- Mathematical Writing (with Paul Roberts)
- Mathematics for the Analysis of Algorithms

- Stanford GraphBase
- The METAFONTbook
- Computers and Typesetting

- MMIXware
- The CWEB System of Structured Documentation
- Literate Programming
- Digital Typography

- 3:16 Bible Texts Illuminated
- Things a Computer Scientist Rarely Talks About

A champion for free access

A champion for free access

- Strongly opposed to granting software patents; wrote a strong letter to both the US Patent and Trademark Office and European Patent Organisation about this.

A champion for free access

- Strongly opposed to granting software patents; wrote a strong letter to both the US Patent and Trademark Office and European Patent Organisation about this.
- TEXwas the first open source big software;

A champion for free access

- Strongly opposed to granting software patents; wrote a strong letter to both the US Patent and Trademark Office and European Patent Organisation about this.
- TEXwas the first open source big software; his codes were widely available

A champion for free access

- Strongly opposed to granting software patents; wrote a strong letter to both the US Patent and Trademark Office and European Patent Organisation about this.
- TEXwas the first open source big software; his codes were widely available
- Wrote a 14 page letter to the board of editors of Journal of Algorithms (JAlgs) about the atrocities of journal pricing.

A champion for free access

- Strongly opposed to granting software patents; wrote a strong letter to both the US Patent and Trademark Office and European Patent Organisation about this.
- TEXwas the first open source big software; his codes were widely available
- Wrote a 14 page letter to the board of editors of Journal of Algorithms (JAlgs) about the atrocities of journal pricing.
- Resulted in
  - the entire board quitting JAlgs and starting an ACM Transactions on Algorithms, and
  - creation of more open access journals

Knuth Prize

Knuth Prize

- Awarded since 1996 and includes an award of $ 5000
- Awarded to individuals for their overall impact in the field.
- Awarded in alternation at the ACM STOC and at the IEEE Symposium on FOCS conferences

Yao (1996) Valiant (1997) Lovasz (1999) Ullman (2000) Papadimitriou (2002) Ajtai (2003) Yannakakis (2005) Lynch (2007) Strassen (2008) Johnson (2010) Kannan (2011) Levin (2012) Miller (2013) Lipton (2014) Babai (2015) Nisan (2016) Goldreich (2017)

Quotations (About him from others!)

Quotations (About him from others!)

- "What is amazing about these three papers of Knuth, and chapter 5 of TACOP is that deep mathematics is done with a very very spare framework - a sort of mathematical minimalism. This shows that Knuth has zoomed in to the fundamental essense of the mathematical idea removing all extraneous distractions. This is something only a master can do."

Quotations (About him from others!)

- "What is amazing about these three papers of Knuth, and chapter 5 of TACOP is that deep mathematics is done with a very very spare framework - a sort of mathematical minimalism. This shows that Knuth has zoomed in to the fundamental essense of the mathematical idea removing all extraneous distractions. This is something only a master can do." Amritanshu Prasad (IMSc)

Quotations (About him from others!)

- "What is amazing about these three papers of Knuth, and chapter 5 of TACOP is that deep mathematics is done with a very very spare framework - a sort of mathematical minimalism. This shows that Knuth has zoomed in to the fundamental essense of the mathematical idea removing all extraneous distractions. This is something only a master can do." Amritanshu Prasad (IMSc)
- Knuth's co-author on his (last?) paper, Fan Chung says "I thought that I was a perfectionist until I met Don Knuth" (http://www.math.ucsd.edu/ fan/paint/math.html)

Quotations of him

Quotations of him

- Talking about the errors he made in his first program
  'My first program taught me a lot about the errors I was going to make in the future and also about how to find errors. ... I try to get things correct. I probably obsess about not making too many mistakes'

Quotations of him

- Talking about the errors he made in his first program
  'My first program taught me a lot about the errors I was going to make in the future and also about how to find errors. ... I try to get things correct. I probably obsess about not making too many mistakes'

- About his love for programming
  'The fact that I have to translate my knowledge of this method into something that the machine is going to understand forces me to make that knowledge crystal-clear in my head. Then I can explain it to somebody else infinitely better.

Quotations of him

- Talking about the errors he made in his first program
  'My first program taught me a lot about the errors I was going to make in the future and also about how to find errors. ... I try to get things correct. I probably obsess about not making too many mistakes'

- About his love for programming
  'The fact that I have to translate my knowledge of this method into something that the machine is going to understand forces me to make that knowledge crystal-clear in my head. Then I can explain it to somebody else infinitely better. The exposition is always better if I have implemented it, ... I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me.'

Respect for Software Enginners

‘I used to think there were different kinds of tasks: writing a paper, writing a book, teaching a class, things like that. I could juggle all of those simultaneously. But software was an order of magnitude harder. I could not do that and still teach a good Stanford class.’

Theory and Practice

The theory I do gives me the vocabulary and the ways to do practical things that can make giant steps instead of small steps when I am doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they are just purely inspired by other theories.

Theory and Practice

The theory I do gives me the vocabulary and the ways to do practical things that can make giant steps instead of small steps when I am doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they are just purely inspired by other theories.
My main message to the theorists is, Your life is only half there unless you also get nurtured by practical work.

Theory and Practice

The theory I do gives me the vocabulary and the ways to do practical things that can make giant steps instead of small steps when I am doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they are just purely inspired by other theories.
My main message to the theorists is, Your life is only half there unless you also get nurtured by practical work.

(Secret motivation ...)

Early programming days

One rather curious thing I've noticed about aesthetic satisfaction is that our pleasure is significantly enhanced when we accomplish something with limited tools.

Early programming days

One rather curious thing I've noticed about aesthetic satisfaction is that our pleasure is significantly enhanced when we accomplish something with limited tools. .... A true programming artist might well resent the introduction of more powerful equipment; today's mass storage devices tend to spoil much of the beauty of our old tape sorting methods.

Early programming days

One rather curious thing I've noticed about aesthetic satisfaction is that our pleasure is significantly enhanced when we accomplish something with limited tools. .... A true programming artist might well resent the introduction of more powerful equipment; today's mass storage devices tend to spoil much of the beauty of our old tape sorting methods.
... We should make use of the idea of limited resources in our own education. We can all benefit by doing occasional "toy" programs, when artificial restrictions are set up, so that we are forced to push our abilities to the limit. We shouldn't live in the lap of luxury all the time, since that tends to make us lethargic. The art of tackling miniproblems with all our energy will sharpen our talents for the real problems, and the experience will help us to get more pleasure from our accomplishments on less restricted equipment.

About rigor in computer science

About rigor in computer science

My work on Metafont introduced me to applications where "correctness" cannot be defined. How do I know, for example, that my program for the letter A produces a correct image? I never will; and I've learned to live with that uncertainty. On the other hand, when I implemented the routines that interpret Metafont specifications and draw the associated bitmaps, there was plenty of room for rigor. The algorithms that go into font rendering are among the most interesting I've ever seen.

About rigor in computer science

My work on Metafont introduced me to applications where "correctness" cannot be defined. How do I know, for example, that my program for the letter A produces a correct image? I never will; and I've learned to live with that uncertainty. On the other hand, when I implemented the routines that interpret Metafont specifications and draw the associated bitmaps, there was plenty of room for rigor. The algorithms that go into font rendering are among the most interesting I've ever seen.

As a user of products from Google and Adobe and other corporations, I know that a tremendous amount of rigor goes into the manipulation of map data, transportation data, pixel data, linguistic data, metadata, and so on. Furthermore, much of that processing is done with distributed and decentralized algorithms that require more rigor than anybody ever thought of in the 60s.

About rigor in computer science

My work on Metafont introduced me to applications where "correctness" cannot be defined. How do I know, for example, that my program for the letter A produces a correct image? I never will; and I've learned to live with that uncertainty. On the other hand, when I implemented the routines that interpret Metafont specifications and draw the associated bitmaps, there was plenty of room for rigor. The algorithms that go into font rendering are among the most interesting I've ever seen.

As a user of products from Google and Adobe and other corporations, I know that a tremendous amount of rigor goes into the manipulation of map data, transportation data, pixel data, linguistic data, metadata, and so on. Furthermore, much of that processing is done with distributed and decentralized algorithms that require more rigor than anybody ever thought of in the 60s. The fact that correctness can't be defined on the "bottom line" should not lull people into thinking that there aren't intermediate levels within every nontrivial system where correctness is crucial.

Advice for researchers ...

Advice for researchers ...

- 'When my books came out, they were not copies of any other books. They always were something that had not been fashionable to do, but they corresponded to my own perception of what ought to be done. Do not just do trendy stuff. If something is really popular, I tend to think: back off. I tell myself and my students to go with their own aesthetics, what they think is important. Dont do what you think other people think you want to do, but what you really want to do yourself. That has been a guiding heuristic for me all the way through.'

Thank You


I am here.