# Ask HN: What is your favorite CS paper?

705 points by lainon 23 hours ago | hide | past | web | 226 comments | favorite

add comment

joaobatalha 19 hours ago [-]

"Reflections on Trusting Trust" by Ken Thompson is one of my favorites.

Most papers by Jon Bentley (e.g. A Sample of Brilliance) are also great reads.

I'm a frequent contributor to Fermat's Library, which posts an annotated paper (CS, Math and Physics mainly) every week. If you are looking for interesting papers to read, I would strongly recommend checking it out - http://fermatslibrary.com/

- Reflections on Trusting Trust (Annotated Version) - http://fermatslibrary.com/s/reflections-on-trusting-trust

- A Sample of Brilliance (Annotated Version) - http://fermatslibrary.com/s/a-sample-of-brilliance

reply

khaledh 8 hours ago [-]

Ken Thompson's "Reflections on Trusting Trust" reminds me of this answer to "What is a coder's worst nightmare?": https://www.quora.com/What-is-a-coders-worst-nightmare/answe...

reply

decentralised 3 hours ago [-]

Joao, espero que nao leves a mal mas a tua versão anotada do Ethereum white paper não fez muito sentido... e um trabalho ainda em curso?

reply

wrinkl3 15 hours ago [-]

The Ken Thompson Hack is a haunting idea. The intelligence agencies almost certainly would've tried to implement it at some point.

reply

exelius 14 hours ago [-]

They have -- this is basically what Stuxnet did. Some of the equation group leaks were even further advanced -- they installed themselves in the hard drive firmware, then hid the sectors where the exploit code was stored *even from the BIOS*.

So point is, it's been done. That's why the Equation Group malware operated undetected for almost a decade (and maybe longer than that). Never underestimate the power of a government -- they can afford to hire and train an army of Ken Thompsons for the price of an aircraft carrier.

reply

exikyut 9 hours ago [-]

> they installed themselves in the hard drive firmware, then hid the sectors where the exploit code was stored even from the *BIOS.*

Where can I read more technical details (such as code analysis) about this?

I've never heard of anything like this hiding for *10 years* before.

reply

mdasen 13 hours ago [-]

You can actually detect the issue in Trusting Trust: https://www.schneier.com/blog/archives/2006/01/countering_tr...

If you have two compilers and one is open source (and you've read the source and happy that it's clean), you can compile that source with both compilers. The output will be different because the two compilers will make different optimizations. However, now you have two binaries of the same compiler and while they aren't the same, their output will be. So you can re-compile the source with both new binaries and you should get a bit-for-bit equivalent output.

reply

nialv7 6 hours ago [-]

You can not. The point is you cannot run the code of a compiler, you have to run the compiled binary. And there's no way to verify if the binary does the same thing as the code when the Ken Thompson Hack is implemented.

reply

djhworld 16 hours ago [-]

This is a really nice site, but the owner should really enable HTTPS if they want people to give their email address over for the newsletter

reply

manigandham 9 hours ago [-]

Why? Emails are not private information.

reply

skypather 13 hours ago [-]

second!

reply

tequila_shot 17 hours ago [-]

http://fermatslibrary.com/ isn't opening. Help?

reply

joaobatalha 17 hours ago [-]

Not sure what happened, it was down, but it seems to be back up now. team@fermatslibrary.com is usually pretty responsive

reply

cs702 19 hours ago [-]

I would never call it my "all-time favorite" (no paper qualifies for that title in my book), but Satoshi Nakamoto's paper, "Bitcoin: A Peer-to-Peer Electronic Cash System" deserves a mention here, because it proposed the first-known solution to the double-spending problem in a masterless peer-to-peer network, with Byzantine fault tolerance (i.e., in a manner resistant to fraudulent nodes attempting to game the rules), via a clever application of proof-of-work:

https://bitcoin.org/bitcoin.pdf

Others in this thread have already mentioned papers or opinionated essays that quickly came to mind, including "Reflections on Trusting Trust" by Ken Thompson, "A Mathematical Theory of Communication" by Claude Shannon (incredibly well-written and easy-to-follow given the subject matter), and "Recursive Functions of Symbolic Expressions and Their Computation by Machine" by John McCarthy.

I would also mention "On Computable Numbers, with an Application to the Entscheidungsproblem" by Alan Turing, "On Formally Undecidable Propositions of Principia Mathematica And Related Systems" by Kurt Gödel, and "The Complexity of Theorem Proving Procedures" by Stephen Cook, but in my view these papers are 'unnecessarily' challenging or time-consuming to read, to the point that I think it's better to read textbooks (or popular works like "Gödel, Escher, and Bach" by Douglas Hofstadter) covering the same topics instead of the original papers. Still, these papers are foundational.

Finally, I think "The Mythical Man-Month" by Fred Brooks, and "Worse is Better" by Richard Gabriel merit inclusion here, given their influence.

This is by no means an exhaustive list. Many -- *many* -- other worthy papers will surely come to mind over the course of the day that I won't have a chance to mention here.

There are many other good recommendations elsewhere in this thread, including papers/essays I have not yet read :-)

reply

oculusthrift 19 hours ago [-]

maybe i'm a little stringent but if if isn't peer reviewed and in a journal, i don't consider it a paper.

reply

joeyspn 18 hours ago [-]

Maybe it didn't appear in journals, but certainly has had much more impact than 99% of the papers in the last decade.

An amazing contrast: "Publish or perish" driven research vs "I don't want the fame, I just want to build something useful and practical".

reply

majewsky 19 hours ago [-]

I guess that, given its impact, it's more peer-reviewed than most published papers.

reply

nicoburns 14 hours ago [-]

I think that this is an important point: that just because something isn't in a journal, doesn't mean that it hasn't been peer reviewed!

reply

nikhizzle 21 hours ago [-]

Without a doubt.

Time, Clocks, and the Ordering of Events in a Distributed System. Leslie Lamport.

http://amturing.acm.org/p558-lamport.pdf

My first introduction to time scales as a partial ordering. Very mind opening.

reply

e12e 20 hours ago [-]

A somewhat lighter take on similar issues that I also enjoyed was:

"Designing croquet's TeaTime: a real-time, temporal environment for active object cooperation", David Reed :

http://dl.acm.org/citation.cfm?id=1094855.1094861

reply

navneet4735 20 hours ago [-]

Exactly I worked my way backwards to this paper while exploring real world distributed systems like Kafka and Zookeeper and it was exceptionally well written paper that explained the basics of building distributed systems

reply

lukateake 19 hours ago [-]

Your link is throwing a 404 for me. I found it here:
https://www.ics.uci.edu/~cs230/reading/time.pdf

reply

nikhizzle 15 hours ago [-]

That is not the paper. It appears to be a summary.

reply

0xf8 21 hours ago [-]

"A Mathematical Theory of Communication" - Claude E. Shannon

http://math.harvard.edu/~ctm/home/text/others/shannon/entrop...

reply

ape4 13 hours ago [-]

Definition of a fairly important term from that paper...

"If the base 2 is used the resulting units may be called binary digits, or more briefly *bits*, a word suggested by J. W. Tukey"

reply

godelmachine 20 hours ago [-]

All time hit paper

reply

rhaps0dy 18 hours ago [-]

I like the papers that describe the OOPS and the Gödel Machine :)

reply

thristian 21 hours ago [-]

Out Of The Tarpit, by Moseley and Marks

https://github.com/papers-we-love/papers-we-love/blob/master...

The first half of the paper is a spot-on critique of so many things that go wrong in the process of designing and implementing large-scale software systems. The second half, where the authors propose a solution, kind of goes off the rails a bit into impracticality... but they definitely point in a promising direction, even if nobody ever uses their concrete suggestions.

reply

agentm 18 hours ago [-]

Project:M36 is an implementation of the proposed design from the "Out of the Tarpit" paper.

https://github.com/agentm/project-m36

reply

nuclx 19 hours ago [-]

Definitely a gem. I loved their accurate comparison of the common programming paradigms, though I have to admit, that I didn't follow their idea of functional relational programming in detail.

reply

akkartik 20 hours ago [-]

Peter Naur, *"Programming as theory building."* (1985)

"…programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts."

http://pages.cs.wisc.edu/~remzi/Naur.pdf https://news.ycombinator.com/item?id=10833278 https://news.ycombinator.com/item?id=7491661

reply

defined 10 hours ago [-]

This really is a paper with deep implications.

One of them, as I understand it, is that in any significant software project, regardless of the volume and quality of the documentation, or quality of the code base, maintainers not involved in building the original project will not be able to build the "theory" correctly in their minds, and will consequently make changes that are clumsy or detrimental. (I'm summarizing, so some important aspects have been skipped).

I see this aspect of the paper as related to the "conceptual integrity" discussed in the Mythical Man-Month.

This paper has long been one of my favorites, and was first brought to my attention when I was reading (IIRC) one of Alistair Cockburn's books. Sadly, few of the people

I shared it with found it interesting.

reply

akkartik 8 hours ago [-]

It's at the core of my software life for several years now: http://akkartik.name/about

reply

KirinDave 18 hours ago [-]

I've been trying to get it frontpaged because, despite it's length, it's perhaps one of the most startling papers of this decade. Sadly, it seems like the HN voting gestalt hasn't decided to upvote a paper that's the CS equivalent of breaking the speed of light:

"Generic Top-down Discrimination for Sorting and Partitioning in Linear Time" ->

http://www.diku.dk/hjemmesider/ansatte/henglein/papers/hengl...

(if you're daunted by an 80 page paper as I am, there is also a talk on it: https://www.youtube.com/watch?v=sz9ZlZIRDAg)

It is possible, with some proper insight and approaches, to sort general datastructures in linear time on modern computing hardware. The speed limit of sort is O(n) with some extra constant cost (often accrued by allocation). It works by decomposing and generalizing something akin to radix sort, leveraging a composable pass of linear discriminators to do the work.

There's a followup paper using this to make a very efficient in-memory database that one could easily generalize under something like kademelia and with care I suspect could make something like a better spark core.

http://www.diku.dk/hjemmesider/ansatte/henglein/papers/hengl...

I keep submitting and talking about this but no one seems to pick up on it. This paper is crazy important and every runtime environment SHOULD be scrambling to get this entire approach well-integrated into their stdlib.

Unsurprisingly, Kmett has already implemented it in Haskell (it generalized neatly under the dual of the applicative+alternative functor): https://hackage.haskell.org/package/discrimination

reply

kraghen 16 hours ago [-]

I'm happy to see this excellent paper mentioned. Fritz Henglein (the author) was my thesis supervisor last year, and I worked on developing some of his ideas further.

In particular, I generalised discrimination (and added a touch of linear algebra) to devise a simple multi-way join algorithm that computes the join of any number of relations in optimal time (in a specific technical sense). Such results have been obtained recently, but only with far more complicated algorithms.

Alas, the fully general proof of optimality eludes us, so nothing has been published yet.

reply

ibdknox 15 hours ago [-]

Is there anything written about your multi-way join algorithm? We've done some really interesting implementations of Generic Join and the like and are always interested in hearing about other potentially better join algorithms.

reply

KirinDave 16 hours ago [-]

I can only do a humble and poor job of explaining this process by analogy to other algorithms. If you've got a better explanation I could rote memorize, I'd be very appreciative.

reply

kraghen 16 hours ago [-]

I can't say I have had much success explaining my thesis clearly to anyone except people from the same department, but I can try to give my understanding of discrimination from an implementor's perspective.

The succinct version is that all (discrete) data can be serialised as bit strings and those bit strings can be sorted in linear time by (say) radix sort.

There are two ideas at work here: that all (discrete) data structures ultimately are composed of primitive types that can be sorted in linear time, and that we should only examine each primitive value once.

However, to be fair it should be mentioned that the analysis uses a different machine model (the RAM model) than is usually employed in sorting (where all comparisons are considered to take constant time, which is arguably an inferior model of real computers if you are sorting e.g. strings of variable length).

To be honest, though, the original paper is so well-written that I have a hard time explaining it better.

reply

learningram 18 hours ago [-]

You really improve your persuasion skills.

>Sadly, it seems like the HN crowd won't upvote a paper that's the CS equivalent of breaking the speed of light:

Comments like this will turn people off the rest of your post.

reply

Denzel 9 hours ago [-]

Different people have different reactions. That line is actually what piqued my interest.

Since AP CompSci in HS it's been hammered into students that any sort based on comparisons has a strict lower bound of O(n*log n).

And sorting is particularly important in search engines, of which I've been working on.

So, an algorithm that drastically improves the speed of sorting would actually open up a few more possibilities to consider.

Thanks for sharing the paper KirinDave, I plan to read it.

reply

KirinDave 17 hours ago [-]

Check my submission history. I tried twice.

I dunno how I "persuade" more.

reply

tom_mellior 16 hours ago [-]

> I dunno how I "persuade" more.

In your original post, the first paragraph made you sound like a crank. You said the paper is like breaking the speed of light, but you don't even mention the *topic* of the paper. I almost stopped reading at that point. Similarly for the sentence towards the end saying that this is "crazy important". People can decide that for themselves. They can decide even better if you put stuff into context.

So to persuade more, for this specific post of yours, I would have suggested to replace the first paragraph by something like: "Here is a surprising paper that shows that you can sort general data structures in linear time!"

reply

KirinDave 16 hours ago [-]

> In your original post, the first paragraph made you sound like a crank. You said the paper is like breaking the speed of light, but you don't even mention the topic of the paper. I almost stopped reading at that point. Similarly for the sentence towards the end saying that this is "crazy important". People can decide that for themselves. They can decide even better if you put stuff into context.

I literally mention the TITLE of the paper, itself precisely explaining its domain & method, directly after the statement and end it with a semicolon.

Let's be real here. The paper is a real dragon of a read. If you're not going to go past a single surprising sentence maybe it was pointless for me to mention it anyways.

> "Here is a surprising paper that shows that you can sort general data structures in linear time!"

I have done this twice, tried different tact twice more, and been downvoted or ignored every time. This is the new me, assuming that folks just don't know how their every artifice of computation is backed by sort. And I suppose... why would they? A great many people simply skip even the basic theory of computation as they join the industry now, and maybe that's okay.

But I say it precisely as I do to generate shock. It *should* be surprising. I've caught people interviewing for principal engineering positions at Google and Amazon off guard with this. It's very, very surprising.

reply

s_gourichon 15 hours ago [-]

> I have done this twice, tried different tact twice more, and been downvoted or ignored every time.

You should have tried this: "This weird trick by a dad sorts in linear time. Check it out!" Proven to work on so many ad-ridden clickbait websites, so why shouldn't it work on HN? ;-)

reply

KirinDave 15 hours ago [-]

"I don't want to live on this planet anymore."

reply

s_gourichon 5 hours ago [-]

I don't know if your comment is humour in reply to my humour, or if you're actually kinda hurt. So let's err on the safe side.

The short comment "weird trick" that you replied to was just a joke intending to yield a smile to readers, including you. Actually I liked the "speed limit" way you previously used to submit the paper on HN.

That said, I skimmed through the article you mention and found it to look serious and instructive (from my experience getting a Ph.D. in computer science / robotics, yet nothing does not guarantee anything) yet needing to allocate a serious time slot for actual understanding. Many people, even on HN, don't upvote due to complexity, yet it was right to submit it. A number of other insightful comments were written in this thread, thanks for them. Also, your ELI5 explanations are interesting.

My current feeling is like: this sort/discriminator stuff is probably valuable, though it will start usage in demanding situations. It may also eventually be used, without their users even knowing, as a private implementation detail of some data structure in high-level languages. Wait and see.

Back to feelings, this planet has some drawbacks but all in all it's worth it. B612 is

too small, we're better here. You can expect good things from HN an similar communities but don't expect too much. Try to refrain from complaining, this feeds the negative part of you and readers as human nature tends to stick bad karma to the ones who complain. Also, when disappointed try to not misattribute causes and favor doubt. Feed the positive part of life.

reply

and0 17 hours ago [-]

Complaining that people aren't paying attention to a sorting paper is kinda weird and makes people take you less seriously.

Try to write a blog post on Medium that breaks it down and submit it that way.

reply

KirinDave 16 hours ago [-]

Why on earth would I write for Medium? Will they pay me?

I think it's weird that the entire industry is not burning a hole in the atmosphere as they run to implement this wherever they can. It's a very big deal.

I suspect the problem is the paper is 80 pages. But I did link to a youtube talk that covers all the core features.

reply

THE_PUN_STOPS 17 hours ago [-]

On the contrary, I'm of the opinion that the GP's line you referenced is nearly the perfect hook for HNers. [1]

[1]: https://xkcd.com/386/

Also, you really improve your forgetting a word skills.

reply

nokcha 15 hours ago [-]

>It is possible, with some proper insight and approaches, to sort general datastructures in linear time on modern computing hardware. The speed limit of sort is O(n) with some extra constant cost (often accrued by allocation).

There is a well-known proof from information theory that the problem of sorting *n* distinct items has a worst-case time complexity of Ω(*n* log *n*) fixed-bitwidth operations: (1) Since each of the *n* items is distinct, the average number of bits needed to encode each item is Ω(log *n*). (2) In the worst case, in order to correctly sort the items, each bit of each item needs to be read. (3) Therefore, sorting the list requires reading Ω(*n* log *n*) bits.

So, I'm not sure how to understand the claim that their algorithm operates in "linear time". Are they saying O(n) operations where each operation operates on O(log n) bits?

[Edit: See below response from kraghen: The time is linear in the total size of the data, not in the number of items. I'll leave this comment up in case anyone has the same misunderstanding that I had.]

reply

kraghen 15 hours ago [-]

Discrimination runs in linear time not in the number of items but in the total size of the data. If you have n items each of size k it takes O(kn). Conventional sorting often assumes that you can compare keys of size k in constant time and therefore gets O(n lg n) but a more honest analysis would yield O(kn log n) for (say) merge sort.

reply

kwillets 14 hours ago [-]

The bound on string sorting is typically written as O(n log n + D), where D is the sum of distinguishing prefixes, ie input length minus some fluff.

Since D >= n log n we already have linearity on input length.

reply

kraghen 13 hours ago [-]

Are you saying that you can sort strings in O(n log n + D) using a conventional sorting algorithm such as merge sort? If so, I don't understand why D would be an additive factor implying that each string is only involved in a constant number of comparisons.

(I wasn't, by the way, only considering strings when discussing variable sized keys -- the beauty of discrimination is that we essentially reduce all sorting problems to string sorting.)

reply

kwillets 11 hours ago [-]

A conventional sorting algorithm such as Multikey Quicksort.

http://people.mpi-inf.mpg.de/~sanders/courses/algdat03/salgd...

reply

kraghen 5 hours ago [-]

Sorry, I was being imprecise. By *conventional* I meant comparison-based sorting functions that are polymorphic in the element type and thus not allowed to examine individual bytes.

Multikey Quicksort indeed looks like a special case of discrimination, exploiting some of the same principles.

reply

nokcha 15 hours ago [-]

Thank you, that cleared up my misunderstanding.

reply

joatmon-snoo 5 hours ago [-]

> There is a well-known proof from information theory [...]

Remember that this proof is talking in terms of the number of *comparisons* necessary to sort n items. The moment you stop comparing data, like in radix sort (or more generally, bucket sort), that all flies out the window.

reply

NewEntryHN 15 hours ago [-]

> Are they saying O(n) operations where each operation operates on O(log n) bits

Yes, as in any other claimed complexity about an algorithm. According to your proof finding the maximum in a list is Ω(n log n), which isn't the commonly used measure.

reply

Steeeve 12 hours ago [-]

I have mixed feelings about this one. It certainly has inspired some interesting discussion. The paper itself is obtuse and difficult to absorb.

The methodology isn't _really_ new, but it isn't used frequently. While they showed it as competitive to commonly used sorting algorithms, situationally it can really shine and show significant performance benefits. I'm surprised they didn't show this in one of the graphs (or I missed it in my brief speed-thru).

I don't really see a future for this in standard libraries. I can totally see this methodology being used in the wild in a variety of systems where time = money or entities are looking for a small proprietary distinguishing edge.

reply

KirinDave 11 hours ago [-]

The follow-up paper I linked is less massive and shows a use case that were I younger I would set out with as a startup.

I'm not sure why stdlibs for some languages shouldn't take this approach though. It's difficult, sure, but so is any new foundational tech. What do you see as the barrier?

reply

Steeeve 11 hours ago [-]

complex to do right, limited usefulness, potential for a great deal of memory usage, lack of a common understanding of proper use cases.

I can totally see it as part of any number of extended libraries.

Just my opinion though. I would be happy to be wrong.

reply

jules 13 hours ago [-]

This is a nice technique in theory, but in practice it isn't so fast, mainly because of the implementation details. Radix sort *is* actually faster than comparison sorts if implemented well.

reply

KirinDave 11 hours ago [-]

Do you have benchmarks of Kmett's Implementation? I'm very curious.

reply

stevenschmatz 17 hours ago [-]

What's the catch? Are there any preconditions on the input required?

reply

kraghen 16 hours ago [-]

All orderings must be specified as a reduction to a primitive order using the fact that if you have an equivalence relation on some type A and a reduction f : B -> A then you have an equivalence on B defined by x = y when f(x) = f(y).

Now, take the rational numbers. For the usual binary comparison we can simply define (a/b) < (c/d) as ad < cb. It's not obvious how to express this as a reduction to a primitive ordering (the answer is to compute the continued fraction).

In fact, I'm not aware of any systematic way of deriving discrimination-suitable orderings from binary predicates -- it might be an open research problem, as far as I am aware.

reply

KirinDave 16 hours ago [-]

> In fact, I'm not aware of any systematic way of deriving discrimination-suitable orderings from binary predicates -- it might be an open research problem, as far as I am aware.

That'd be an even more remarkable discovery in light of the stuff you worked on though, wouldn't it?

reply

kraghen 16 hours ago [-]

It might, I never really discussed this aspect with Fritz! For my thesis I was mostly focussed on applications to database queries, and I never encountered any concrete examples of orderings that couldn't be dealt with in an obvious way using discrimination based sorting.

reply

KirinDave 17 hours ago [-]

No. The catch is that it's hard to write it in a way that doesn't require very fiddly local mutation or explosive allocation.

The other catch is that no one has demonstrated that you can do it without a good static type system. It shouldn't be impossible, but there are a lot of

challenges in an already challenging algorithm.

reply

gbacon 17 hours ago [-]

What are the constants hiding inside that $\mathcal{O}(n)$?

reply

KirinDave 16 hours ago [-]

They can be bad. But so was merge sort in its naive implementation and folks worked that out. Radix sort sees a lot of use in the real world and it saves a lot of energy.

reply

csomar 18 hours ago [-]

Given that I'm an ignorant. Can you ELI5 like what is the impact of this? What does it change? What can you do with it on practical terms?

reply

munin 17 hours ago [-]

Imagine that you make your living in a somewhat unorthodox but honest way: you live in a tent outside of the castle and every morning, as the king walks into the castle, he hands you a deck of cards. "I've just played cards last night," he says "and I'm going to again tonight, and I'd like these cards ordered and sorted by suite and number by this evening." Then the king goes into his castle and does whatever it is that kings do in castles all day.

So, you take this deck of cards, which is in any old order, as the king just played with it last night, and you set about sorting it. Now, you've been doing this a long time, and your methodology has improved over the years. When you started, you would just lay the cards out in the order the king gave them to you, and then start at the beginning of the line and ask "is this card greater than the next?" and move the two cards in response to the answer so that the two are in order. You would continue this process, repeating from the beginning of the line, until you made a complete pass through the line without making any changes. Sometimes, this took you all day, sometimes you would even make the king wait for you to finish, and that made him very angry, which is very bad.

Your sorting process is compounded by the limitation of your ability to read and compare the value of two cards, because you don't really know how to read or add, it takes you about ten seconds to decide if a card is greater than or less than another card. For a 52 card deck, this means your original sorting method would require 52 times 52 comparisons, or, in real time, seven and a half hours of non-stop work, leaving you no time to do anything else.

Over the years of doing this job for the king you discovered some shortcuts that would take less time but still produce a sorted deck, despite your limited ability to read and understand the values of the cards. While that still takes ten seconds, your new deck sorting approaches require 52 times 1.7 comparisons, or in real time, about fifteen minutes. This is much, much, better than before, but it would of course be better still if it took even less time, as you have discovered that you can now use this extra time to sort cards for all the other members of the court.

One day a traveling wizard observes you sorting cards and says "you know, this method that you have for sorting cards is quite clever but what if I were to tell you that you could sort these cards in 8.6 minutes flat?" This is about half the time it takes you right now, meaning that you could double the number of decks you sort in a day, doubling your income. You are very interested! "Tell me more," you tell the wizard. "Here, read this paper" the wizard replies, and they give you the paper GP linked.

reply

posterboy 1 hour ago [-]

quite the catch at the end. The wizard probably talked about teaching a man to fish, too.

reply

KirinDave 17 hours ago [-]

ELI5 Edition: Basically everything uses sort (either to rank things or to set up search), so improvements to sort improve basically everything.

Explaining it in further detail like you're a fellow member of our industry:

Edge computing advances are pretty important right now, since the amount of data we're working with on the edge of the network is growing very quickly. Advances in how we compute backpropagation (essentialy using a right-recursive process for the Chain Rule) mean that we can do machine learning on 2015-level phone gpus, rather than 2015-level desktop GPUs.

This advance hints at a promise of similar gains. With care, you can sort much faster. What's more, your sorts and your merge of decomposed sorts is roughly the same cost now. And multiple, layered sorts don't require custom logic or clever functional programming (at the edge developer's model) to compose.

Essentially you pick what fields in complex (maybe even nested) data structures to sort by, in what order, and the system makes a sort.

Why does this matter? Well, it will make nearly all data structures Just Faster; many of them secretly rely on ordered and sorted arrays. It makes it easier for distributed systems authors to make queryable systems (it's pretty easy to write a discriminator or specify a discriminator function remotely and then use that on the fly, as opposed to a comparator).

One place in the client-software world where it can have big impact is offline webapps. Right now, it's almost always cheaper to call out to a remote datastore rather than use a local datastore even if you're using sqlite under the covers because of the cost of synchronizing sqlite's data model. A discriminator-based approach would let you do complex queries of data in memory, but that data could still be themselves commutative data types that are coalescing data out of a data stream (or multiple data streams) remotely.

It's also worth noting that another really cool paper from this year improving on the HAMT trie (https://michael.steindorfer.name/publications/phd-thesis-eff...) could also benefit from this approach, which means all of us using immutable data structures can continue to do so with MUCH better performance characteristics but continued thread safety.

reply

KirinDave 17 hours ago [-]

I just wanted to give you a taste of how this works in an ungeneralized way before I went. Linear time algorithms are fast and feel very different. I want to stress this is not directly related to the technique in the paper, but is in the same "family" of algorithms, and is very easy to understand.

Imagine you're doing the classic "write an algorithm to determine if one word is the anagram of another". The classic solution is to sort and compare the strings to be checked.

We can do this pretty elegantly for strings without using quicksort. It goes like this: allocate a block of 26 bytes. Each byte is a counter for a letter in that position (0 -> A, 1 -> B, ...). Now sweep across the string, and each time you see a letter, increment that number. If you really care hard, you can go crazy with SIMD and other clever tricks here.

Once you're done this for 2 strings, you need only compare the two regions for equality (a constant time operation).

The worst case, average case, and minimum running time of this algorithm is O(len_word_1+len_word_2)+c, and because we can safely make assumptions about the length of the strings (no word in anym ISO-Latin-1 encoding exceeds 255 of any character), we can do it fairly compactly. This is MUCH faster and can be done with perfect memory safety (which is to say, it is optimal with mutability but all operations are commutative so they may be subdivided, done in any order, and recombined at will).

Try implementing this. It's really, really fast on modern hardware. Especially if you're working it out for a full /usr/share/dict/words file on a normal _nix/bsd installation.

We can even see that composability feature in our current problem! Imagine we have have that big dictionary file and we want to find ALL the anagrams in it. Imagine we start cutting the above technique in half and computing the byte vector for every word in the dictionary (which is O(n) time). If we sort

THESE vectors, we could just pick out all the equal values and say "There are the anagram groups", right?

If we were to insert them into some very wide trie (as is the style these days), that'd be O(Numwords * log(numwords)), which is really just sorting again. We'd do it in minimum 2 passes with O(n log n) cost. But if we have *composable discriminators* we could do this in one big pass with O(n) cost! Our constant factors would grow because we're dealing with more memory. We could then not only sweep across the sorted list to extract groups (note the symmetry with the ABC counter approach above), but we could also pretend it's a tree structure (start at the length/2 element in the list and pretend its a binary tree) and search for new items inside the block, so we could check for new word anagrams subsequently in O(log n) time.

The paper I linked talked about generalizing this idea (a special case of radix sort) and making it composable.

reply

flavio81 20 hours ago [-]

Automated Distributed Execution of LLVM code using SQL JIT Compilation

As collected by the SIGBOVIK group:

http://sigbovik.org/2017/proceedings.pdf

Quote:

"Following the popularity of MapReduce, a whole ecosystem of Apache Incubator Projects has emerged that all solve the same problem. Famous examples include Apache Hadoop, Apache Spark, Apache Pikachu, Apache Pig, German Spark and Apache Hive [1]. However, these have proven to be unusable because they require the user to write code in Java. Another solution to distributed programming has been proposed by Microsoft with their innovative Excel system. In large companies, distributed execution can be achieved using Microsoft Excel by having hundreds of people all sitting on their own machine working with Excel spreadsheets. These hundreds of people e combined can easily do the work of a single database server."

PS: This thread is great, i'm bookmarking because here there are good (serious) papers.

reply

jlisam13 20 hours ago [-]

Apache Pikachu got me

reply

rhaps0dy 18 hours ago [-]

The alliteration is real

reply

godelmachine 20 hours ago [-]

Me, too bookmarking!

reply

andars 19 hours ago [-]

I'll take a broad interpretation of 'CS' and throw out a couple of personal highlights.

C. Shannon, "A Symbolic Analysis of Relay and Switching Circuits" (1940):
https://dspace.mit.edu/bitstream/handle/1721.1/11173/3454142...

Shannon's master's thesis, which introduces boolean algebra to the field of digital circuit design.

R.W. Hamming, "Error Detecting and Error Correcting Codes" (1950):
https://ia801903.us.archive.org/1/items/bstj29-2-147/bstj29-...

In Hamming's own words: "Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?"

J.T. Kajiya, "The Rendering Equation" (1986):
http://cseweb.ucsd.edu/~ravir/274/15/papers/p143-kajiya.pdf

Kajiya introduces the integral rendering equation, which is the basis for most current techniques of physically based rendering.

reply

gregors 21 hours ago [-]

"Reflections on Trusting Trust" - Ken Thompson

https://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thomp...

reply

dkaoster 20 hours ago [-]

Seconded.

reply

noobtube 20 hours ago [-]

this is the bomb.

reply

jasode 21 hours ago [-]

"The Limits of Correctness" (1985) by Bryan Cantwell Smith:
https://www.student.cs.uwaterloo.ca/~cs492/11public_html/p18...

I know Thompson's "Reflections on Trust" and Shannon's "Communication" papers are more famous but I believe BCS's "Correctness" paper has more immediate relevance to a wider population of programmers.

For example, I don't believe Ethereum's creator, Vitalik Buterin, is familiar with it because if he was, he would have realized that *"code is law"* is not possible and therefore he would have predicted the DAO hack and subsequent fork/reversal to undo the code.

Seriously, if you read BCS's paper *and generalize its lessons learned*, you will see that the DAO hack and its reversal as *inevitable*.

reply

LeanderK 20 hours ago [-]

i enjoyed skimming the paper, but i don't agree to your conclusions. It reminds me of an recent discussion i had about Dependent Types in Haskell, why not just check for termination instead of asserting it (I know about the halting problem)?

I think for many applications there is no binary answer, it's not just a good or bad idea. The question is how good can we get and is it any better than the state of the art? There are theoretical limits, but the interesting part is whether there exists a practical approximation. I don't believe in a fundamental difference between us and computers, i think everything we can reason about should be possible to algorithmically reason about. I think smart-contracts are a fundamental improvement over "non-code as law", i really believe in them. They are reproducible and exact. But it's a shame that solidity is so badly engineered, because they it is really hard to prove anything in it. I think they did the exact opposite of what would be the right language. I understand the reasoning behind "the limits of correctness", but does this means that proving anything is meaningless?

I would expect most contracts to be stupidly simpel, at least to a machine, with simpel properties that need to be proven comparable to testing Haskell with quickcheck. And i believe they are an improvement over "non-code as law", even if not provably correct.

The problem with bugs and smart-contracts is interesting. But implementing smart-contracts does not mean automating the judge.

reply

catnaroek 21 hours ago [-]

> you will see that the DAO hack and its reversal as inevitable.

Honest naïve question. What's the proof?

reply

throwawayjava 20 hours ago [-]

FWIW I think this is a very fair question. Parent's post veers a bit further toward defeatism than I think Smith's paper advocates for or justifies. In particular, *of course* the DAO hack wasn't inevitable -- a more careful programmer could've foreseen and prevented that attack and whole other classes of attack.

reply

tom_mellior 20 hours ago [-]

Part of the linked paper's point is that because computer systems involve many "levels of failure", even a more careful programmer cannot usually rule out every class of programs. The DAO hack, yes, possibly.

But, for example, people have also lost money due to bugs in the Solidity compiler: https://np.reddit.com/r/ethtrader/comments/5foa5p/daily_disc... How many "more careful" Ethereum programmers *also* check the compiler for correctness?

Another paper in this vein is James Fetzer's "Program Verification: The Very Idea". http://lore.ua.ac.be/Teaching/SSPEC2LIC/critique2.pdf

reply

throwawayjava 19 hours ago [-]

*> Part of the linked paper's point is that because computer systems involve many "levels of failure", even a more careful programmer cannot usually rule out every class of programs.*

But -- and this is the crucial point -- that doesn't mean we shouldn't strive to be better than we are now.

The impossibility of perfectly modeling the world hasn't prevented us from making enormous progress on software safety and security over the past 30 odd years. Today, if you care to, you can easily write code that is free of buffer overflows and command injection attacks. In the 00's SQL injection attacks were extremely easy to find; now they're comparatively rare.

Smith's paper tells us that code-as-law is probably a bad idea. But it is not -- and wasn't intended to be -- an indict of static analysis or model-based engineering more generally. Every structural engineer knows the difference between a bridge and a model of a bridge; a paper pointing out the difference without substantively critiquing the practical usefulness of a particular type of model would probably elicit eye-rolls. I'm not sure why these mundane observations receive such attention in computer science. Maybe because with software the model and the system look so similar.

But to be sure, the impossibility of codifying human morality is a pretty lame excuse for failing to use static analysis tools or better languages or quality frameworks to prevent known classes of attacks. So I doubt that's what Smith is advocating.

*> How many "more careful" Ethereum programmers also check the compiler for correctness?*

Yes, we should *obviously* check compilers for correctness, and we're making slow but sure progress toward a world where our critical infrastructure comes with strong -- of course, never perfect -- correctness guarantees.

reply

tom_mellior 16 hours ago [-]

> But -- and this is the crucial point -- that doesn't mean we shouldn't strive to be better than we are now.

Agreed! And I also agree that we are really making progress. But we're far from a world where people can crank out provably correct code (let alone the *proofs*).

> I'm not sure why these mundane observations receive such attention in computer science. Maybe because with software the model and the system look so similar.

Excellent point, and yes, I think that is the problem. Modeling the world in code is not much different from... er... modeling the world in code :-)

reply

catnaroek 19 hours ago [-]

> Part of the linked paper's point is that because computer systems involve many "levels of failure", even a more careful programmer cannot usually rule out every class of programs.

The programmer only has to comply with the specification. The specification is a finite syntactic entity. If the specification doesn't capture what the user really wants, or compliance is undecidable, or <insert problem beyond the programmer's control here>, then the one at fault is the specification writer, not the programmer.

reply

KirinDave 14 hours ago [-]

Given how bad the tooling is for Eth right now, it probably was inevitable. Their tooling isn't just a mess semantically and syntactically, it's was riddled with double-operation bugs.

reply

agentultra 20 hours ago [-]

Most of my favourites have already been listed but one I found particularly interesting was Von Neumann's *Theory of Self-Reproducing Automata* [0].

[0] http://cba.mit.edu/events/03.11.ASE/docs/VonNeumann.pdf

reply

yaseer 20 hours ago [-]

+1 to this. Von Neumann was well ahead of his time with his automata ideas. I found in interesting how both Von Neumann and Turing turned to analyse biological systems later in their careers.

reply

ehudla 14 hours ago [-]

Some of you might be interested in an essay I wrote about von Neumann and Norbert Wiener that gives some of the background and context of this work.

http://www.ehudlamm.com/outsiders.pdf

reply

tksfz 17 hours ago [-]

Purely Functional Data Structures by Chris Okasaki - https://www.cs.cmu.edu/~rwh/theses/okasaki.pdf

Can programming be liberated from the von Neumann style? - John Backus's Turing lecture - http://dl.acm.org/citation.cfm?id=1283933

reply

chadash 21 hours ago [-]

It might be a cliche one to pick, but I really really really enjoy Alan Turing's "Computing Machinery and Intelligence"[1]. This paper straddles the line between CS and philosophy, but I think it's an important read for anyone in either field. And a bonus is that it's very well-written and readable.

[1] https://www.csee.umbc.edu/courses/471/papers/turing.pdf

reply

AndrewOMartin 19 hours ago [-]

Thanks for picking that one, that means I don't have to choose between it and my joint favourite, the snappily titled "Why Heideggerian AI Failed and how Fixing it would Require making it more Heideggerian".

Where Turing's paper put us on the journey towards AI, 57 years later Dreyfus points out how the direction chosen is not only wrong, but hopelessly misguided.

[quote] As I studied the RAND papers and memos, I found to my surprise that, far from replacing philosophy, the pioneers in CS and AI had learned a lot, directly and indirectly from the philosophers. They had taken over Hobbes' claim that reasoning was calculating, Descartes' mental representations, Leibniz's idea of a "universal characteristic" - a set of primitives in which all knowledge could be expressed, -- Kant's claim that concepts were rules, Frege's formalization of such rules, and Wittgenstein's postulation of logical atoms in his Tractatus. In short, without realizing

it, AI researchers were hard at work turning rationalist philosophy into a research program. [endquote]

http://leidlmair.at/doc/WhyHeideggerianAIFailed.pdf

reply

icc97 19 hours ago [-]

Given the far reaching importance of the Turing test, it's amazing how readable and understandable the paper is.

reply

Retra 11 hours ago [-]

I really don't see how anyone would take that as a given.

reply

twoodfin 20 hours ago [-]

Cheating a little, but the collected Self papers are what I'd bring to a desert island:

https://www.cs.ucsb.edu/~urs/oocsb/self/papers/papers.html

reply

zachsnow 20 hours ago [-]

"Self: the power of simplicity" really lives up to its name!

reply

emidln 21 hours ago [-]

A bit cliche for HN, but I really enjoyed *RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MACHINE (Part I)* by John McCarthy[0]. It was accessible to someone whose background at the time was not CS and convinced me of the beauty of CS -- and lisp.

[0] - http://www-formal.stanford.edu/jmc/recursive.html

reply

irfansharif 19 hours ago [-]

Diego Ongaro's Raft paper[1]. Perhaps this only speaks to my experience as a student but having surveyed some of the other papers in the domain (paxos[2] in its many variants: generalized paxos[3], fpaxos[4], epaxos[5], qleases[6]), I'm glad the author expended the effort he did in making Raft as understandable (relatively) as it is.

[1]: https://raft.github.io/raft.pdf

[2]: https://www.microsoft.com/en-us/research/wp-content/uploads/...

[3]: https://www.microsoft.com/en-us/research/wp-content/uploads/...

[4]: https://www.microsoft.com/en-us/research/wp-content/uploads/...

[5]: https://www.cs.cmu.edu/~dga/papers/epaxos-sosp2013.pdf

[6]: https://www.cs.cmu.edu/~dga/papers/leases-socc2014.pdf

reply

brad0 21 hours ago [-]

Kademlia, a P2P distributed hash table. DHTs are very complex from the outside but very simple once you understand the building blocks.

https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia...

reply

zzzcpan 18 hours ago [-]

Hmm, I think it's important to understand Chord DHT first and only after that move on to Kademlia, then S/Kademlia and so on.

reply

CobrastanJorji 17 hours ago [-]

Yao's minimax principle. It's not a very exciting read or a very exciting conclusion compared to some of these other papers, but it's still interesting, and the conclusion has been practically useful to me a small handful of times.

It concerns randomized algorithms, which are algorithms that try to overcome worst case performance by randomizing their behavior, so that a malicious user can't know which input will be the worst case input this time.

The principle states that the expected cost of a randomized algorithm on a single input is no better or worse than the cost of a deterministic algorithm with random input.

Yao proves this is the case by constructing two zero sum games based around the algorithms' running times and then using game theory (specifically von Neumann's minimax theorem) to show that the two approaches are equivalent. It's a really neat approach!

reply

dvirsky 20 hours ago [-]

The Anatomy of a Large-Scale Hypertextual Web Search Engine, by Brin and Page.

Not only for the historical value of changing the world, and for the fact that it's very interesting and readable; It has personal value to me: the first CS paper I've ever read and it inspired me and changed the course of my life, literally.

Also, it has some very amusingly naive (in hindsight) stuff in it, like: "Google does not have any optimizations such as query caching, subindices on common terms, and other common optimizations. We intend to speed up Google considerably through distribution and hardware, software, and algorithmic improvements. Our target is to be able to handle several hundred queries per second"

http://infolab.stanford.edu/~backrub/google.html

reply

pmiller2 19 hours ago [-]

I was hoping this one would show up. It's a startlingly simple paper, which both made the idea easy to implement (PageRank algorithm can be implemented in around 10 lines of Python, give or take), and easy to game (*viz.* the rise of link farms). Recommended for anyone with the prerequisite 1 semester of linear algebra or equivalent.

reply

romaniv 20 hours ago [-]

I don't have a favorite research paper, but there is a long Ph.D. thesis I've recently read in its entirety and found a lot of interesting ideas:

Programming with Agents: http://alumni.media.mit.edu/~mt/thesis/mt-thesis-Contents.ht...

Here is a short paper with a clear description of an ingenious idea.

Engineered Robustness by Controlled Hallucination: http://web.mit.edu/jakebeal/www/Publications/NIAI-2008.pdf

I like the simplicity of it. Most CS researches seem to be afraid of describing things that are simple, even if those things are non-obviosu and valuable.

reply

vaibhavsagar 19 hours ago [-]

There are a ton of fantastic Haskell papers, but if I had to pick one this would be it. It reconciles the pure and lazy functional nature of Haskell with the strict and often messy demands of the real world:

State in Haskell. John Launchbury and Simon L. Peyton Jones

https://www.microsoft.com/en-us/research/wp-content/uploads/...

reply

amelius 16 hours ago [-]

I like Haskell papers and books but they often reference the "Core" language, for which an accessible implementation seems to be lacking, which is a missed opportunity, imho. Yes, I know it is part of GHC, but it is buried under several layers of undocumented code. GHC could have been much more open to research if they modularized and documented everything more thoroughly.

reply

vaibhavsagar 9 hours ago [-]

I think SPJ would agree with you. Have you seen https://www.youtube.com/watch?v=uR_VzYxvbxg?

reply

bra-ket 19 hours ago [-]

Kanerva's Sparse Distributed Memory is quite remarkable: https://en.wikipedia.org/wiki/Sparse_distributed_memory

it's a book though: https://www.amazon.com/Sparse-Distributed-Memory-MIT-Press/d...

reply

btilly 17 hours ago [-]

*As We May Think* https://www.theatlantic.com/magazine/archive/1945/07/as-we-m...

This paper, written during WW II (!) by someone who had around to 20 years of computing experience at that time (!!) introduced the world to the ideas like hypertext, and citation indexes. Google's PageRank algorithm can be seen as a recombining of ideas from this paper.

This is worth reading to see how much was understood how early.

reply

filereaper 17 hours ago [-]

Some old ones:

Jeffrey Ullman & John Hopcroft: Formal languages and their relation to automata [0]

Ted Codd: A relational model of data for large shared data banks [1]

C.A.R Hoare: Communicating Sequential Processes [2]

[0]http://dl.acm.org/citation.cfm?id=1096945

[1]http://dl.acm.org/citation.cfm?id=362685

[2]http://www.usingcsp.com/cspbook.pdf

reply

zzzcpan 18 hours ago [-]

Worth mentioning Joe Armstrong's "Making reliable distributed systems in the presence of sodware errors" [1].

[1] http://erlang.org/download/armstrong_thesis_2003.pdf

reply

whatversclever 18 hours ago [-]

I was hoping your title wasn't a typo

reply

gens 19 hours ago [-]

"Communicating Sequential Processes" by Tony Hoare

http://www.usingcsp.com/

I read it multiple times and still don't quite understand it all.

There are more great papers i read but this one comes back to mind more often then others.

reply

microbie 21 hours ago [-]

Dijkstra's shortest path algorithm in "A Note on Two Problems in Connexion with Graphs" http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.... Pure, mathematical and a great impact on both how to prove and define algorithms as well as the problem itself.

reply

exelius 21 hours ago [-]

The Mythical Man-Month

https://www.cs.drexel.edu/~yfcai/CS451/RequiredReadings/Myth...

reply

kabdib 19 hours ago [-]

Favorite quote: "I gave my manager two copies of this book so he could read it twice as fast."

reply

monocasa 20 hours ago [-]

And the paper that inspired it, Melvin Conway's "How Do Committees Invent?"

http://www.melconway.com/Home/Committees_Paper.html

reply

exelius 14 hours ago [-]

This is great -- I never knew about this paper but it reinforces a lot of things I learned. Namely, if you want to know how a company *really* works, ignore the org charts and map the systems architecture (a corollary to the author's thesis).

That it still holds true almost 50 years later is pretty amazing.

reply

1001101 21 hours ago [-]

New Directions in Cryptography - Diffie + Hellman

https://www-ee.stanford.edu/~hellman/publications/24.pdf

reply

larkeith 19 hours ago [-]

The Night Watch by James Mickens is always a good read:

https://www.usenix.org/system/files/1311_05-08_mickens.pdf

reply

xenophonf 18 hours ago [-]

That's a good one, but I think "This World of Ours", Mickens' treatise on the practical realities of operational security (specifically keying material handling), should take the top spot:

https://www.usenix.org/system/files/1401_08-12_mickens.pdf

reply

larkeith 18 hours ago [-]

Ooh, I'll have to read that next time I have a few minutes. Thabks for the link!

reply

Analemma_ 14 hours ago [-]

The nice thing about this one is that it actually does have some important kernels of truth amid all the hilarity. Especially this bit about threat models:

> Basically, you're either dealing with Mossad or not-Mossad. If your adversary is not-Mossad, then you'll probably be fine if you pick a good password and don't respond to emails from ChEaPestPAiNPi11s@ virus-basket.biz.ru. If your adversary is the Mossad, YOU'RE GONNA DIE AND THERE'S NOTHING THAT YOU CAN DO ABOUT IT

More security researchers need to learn about that.

reply

godelmachine 6 hours ago [-]

I read The Slow Winter by James Mickens. Reading that brought a revolutionary change in my thinking. Have bookmarked his Harvard profile, just in case he publishes any more articles.

reply

mayank 19 hours ago [-]

The Flajolet-Martin paper on counting unique items in an infinite stream with constant space [1]: a great, well-written introduction to streaming algorithms that triggered my first "aha" moment in the field. You never forget your first.

[1] http://algo.inria.fr/flajolet/Publications/FlMa85.pdf

reply

wsxiaoys 21 hours ago [-]

Cheney on the MTA http://home.pipeline.com/~hbaker1/CheneyMTA.html

Full tail recursion scheme implementation by never "return" in C

reply

twoodfin 20 hours ago [-]

Such a great hack. Not only tail recursion but a complete GC implementation in (almost) platform-independent generated C code.

reply

atilimcetin 18 hours ago [-]

Not a single paper but Eric Veach's Ph.D. thesis 'Robust Monte Carlo Methods for Light Transport Simulation' - http://graphics.stanford.edu/papers/veach_thesis/

reply

ChuckMcM 20 hours ago [-]

I've always enjoyed Finseth's Thesis on text editing, "A cookbook for an EMACS", which he turned into a book: https://www.finseth.com/craft/ and is available in epub form for free.

reply

coherentpony 18 hours ago [-]

An Algorithm for the Machine Calculation of Complex Fourier Series

James W. Cooley and John W. Tukey

Mathematics of Computation Vol. 19, No. 90 (Apr., 1965), pp. 297-301

https://www.jstor.org/stable/2003354

reply

jacobolus 17 hours ago [-]

Apparently this FFT was first discovered by Gauss (1805). http://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf

As a follow-up, let me recommend Püschel & Moura (2006) “Algebraic Signal Processing Theory”, https://arxiv.org/pdf/cs/0612077.pdf and Püschel’s other papers about similar topics.

Or in a different direction, DJB (2008), “Fast multiplication and its applications”, http://cr.yp.to/lineartime/multapps-20080515.pdf

reply

andars 16 hours ago [-]

Interesting to note that Fourier's two works were published in 1807 and 1822, so that work by Gauss also predates Fourier.

reply

protomyth 14 hours ago [-]

I would say An Agent-Oriented Programming by Yoav Shoham. It certainly set my mind going and made me think about how programs could be organized. I still think, agents, systems of agents, and mobile agent code has a place in computing. Even though some form of RPC over HTTP won over mobile code, I look at the spinning up of VMs and cannot help but think that agents have a place. Combined with the tuple space stuff from Yale, I still see a powerful way to go forward.

1) 1990 http://cife.stanford.edu/node/599

2) 1993 http://faculty.cs.tamu.edu/ioerger/cs631-fall05/AOP.pdf

reply

nrjames 20 hours ago [-]

Mine is "Image Quilting for Texture Synthesis and Transfer" by Efros and Freeman. It's simple enough to implement as a personal project and has some nice visual output. Plus, Wang tiles are cool and it's fun to learn more about them.

https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/p...

reply

BugsJustFindMe 18 hours ago [-]

On the same subject, I prefer the concurrently published "Image Analogies" paper ( http://www.mrl.nyu.edu/projects/image-analogies/ ) because their described technique, while slow, just makes so much sense.

reply

taeric 19 hours ago [-]

"Dancing Links" by Knuth (http://www-cs-faculty.stanford.edu/~knuth/papers/dancing-col...) is still one of my favorites for my actually having understood it. :) I wish I had found it back in grade school. (Though I suspect I wouldn't have understood it, then.)

reply

random_comment 20 hours ago [-]

Depixelizing Pixel Art

http://johanneskopf.de/publications/pixelart/paper/pixel.pdf

I think this paper is very cute and also technically interesting.

reply

arde 18 hours ago [-]

All of the classic papers I can think of have already been mentioned, but even though it's too recent to pass judgment a new contender may well be "Deep Learning and Quantum Entanglement: Fundamental Connections with Implications to Network Design" - https://arxiv.org/abs/1704.01552

reply

popcorncolonel 18 hours ago [-]

Damn, did not expect to see this paper on here. When I read it, I wasn't thinking it could be listed under the "all-time favorite CS papers", but the connections are quite interesting and (retrospectively) intuitive.

I'll be interested to see where/(if) it gets published.

reply

bluedino 20 hours ago [-]

"A Method for the Construction of Minimum-Redundancy Codes"

https://www.ic.tu-berlin.de/fileadmin/fg121/Source-Coding_WS...

I'm not sure if it was the fact that I was just a kid when I read it, but it was just so obvious and simple but so complicated and amazing at the same time.

reply

bootsz 20 hours ago [-]

End-To-End Arguments in System Design

http://web.mit.edu/Saltzer/www/publications/endtoend/endtoen...

reply

lawn 20 hours ago [-]

Bitcoin: A Peer-to-Peer Electronic Cash System

https://www.bitcoin.com/bitcoin.pdf

reply

akkartik 20 hours ago [-]

Hofstadter, D. R. and Mitchell, M. (1995). *"The Copycat project: A model of mental fluidity and analogy-making."* Chapter 5 in D. R. Hofstadter, Fluid Concepts and Creative Analogies.

http://web.cecs.pdx.edu/~mm/CopycatChapter.html

"Copycat is a model of analogy making and human cognition based on the concept of the parallel terraced scan, developed in 1988 by Douglas Hofstadter, Melanie Mitchell, and others at the Center for Research on Concepts and Cognition, Indiana University Bloomington. Copycat produces answers to such problems as "abc is to abd as ijk is to what?" (abc:abd :: ijk:?). Hofstadter and Mitchell consider analogy making as the core of high-level cognition, or high-level perception, as Hofstadter calls it, basic to recognition and categorization. High-level perception emerges from the spreading activity of many independent processes, called codelets, running in parallel, competing or cooperating. They

create and destroy temporary perceptual constructs, probabilistically trying out variations to eventually produce an answer. The codelets rely on an associative network, slipnet, built on pre-programmed concepts and their associations (a long-term memory). The changing activation levels of the concepts make a conceptual overlap with neighboring concepts." -- https://en.wikipedia.org/wiki/Copycat_(software)

https://cogsci.indiana.edu/copycat.html

reply

uvatbc 6 hours ago [-]

One of my all time favorites has been the Jefferey Mogul paper on Receive Livelock: https://pdos.csail.mit.edu/6.828/2008/readings/mogul96usenix...

I read it first as a normal CS paper, but later started seeing it as a commentary on an extremely busy work life.

Right there in the first paragraph: "... receive livelock, in which the system spends all its time processing interrupts, to the exclusion of other tasks..."

Does this remind you of anything?

reply

otakucode 20 hours ago [-]

Admittedly a good portion of my appreciation is due to the title alone, but the paper and contents itself are very good as well:

'The Geometry of Innocent Flesh on the Bone: Return-into-libc without function calls' by Hovav Shacham

http://cseweb.ucsd.edu/~hovav/dist/geometry.pdf

reply

p4bl0 12 hours ago [-]

It's not exactly a paper but I really liked "The Limits of Mathematics" by Chaitin. I wrote a blogpost about it a few years back (https://shebang.ws/the-limits-of-mathematics.html), I already submitted it to HN (https://news.ycombinator.com/item?id=1725936).

reply

kwindla 11 hours ago [-]

Alexia Massalin's 1992 PhD thesis describing the Synthesis Operating System.

```
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.4871&rep=rep1&type=pdf
```

Here's Valerie Aurora's description of Synthesis:

... a completely lock-free operating system optimized using run-time code generation, written from scratch in assembly running on a homemade two-CPU SMP with a two-word compare-and-swap instruction—you know, nothing fancy.

Which (necessarily) undersells by a very large margin just how impressive, innovative, and interesting this thesis is.

If you're interested in operating systems, or compilers, or concurrency, or data structures, or real-time programming, or benchmarking, or optimization, you should read this thesis. Twenty-five years after it was published, it still provides a wealth of general inspiration and specific food for thought. It's also clearly and elegantly written. And, as a final bonus, it's a snapshot from an era in which Sony made workstations and shipped its own, proprietary, version of Unix. Good times.

reply

xixixao 21 hours ago [-]

Notation as a Tool of Thought, Kenneth E. Iverson

http://www.jsoftware.com/papers/tot.htm

reply

> evanb 21 hours ago [-]
>
> Also along those lines: Two notes on notation, Knuth, https://arxiv.org/abs/math/9205211
>
> reply

beefman 19 hours ago [-]

Backus - A Functional Style and Its Algebra of Programs

https://www.cs.ucf.edu/~dcm/Teaching/COT4810-Fall%202012/Lit...

reply

mooneater 18 hours ago [-]

"On the criteria to be used in decomposing systems into modules" by David Parnas, 1972, the seminal paper where he brings forward the key ideas that would later be called cohesion and coupling.

https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/crit...

Why it was important: you can't build big complex systems without these principles.

Some people say he was instrumental in stopping the Star Wars program, he argued it would be impossible to test outside of war (and therefore doomed).

reply

mdhughes 20 hours ago [-]

Worse is Better http://wiki.c2.com/?WorseIsBetter

reply

GeorgeTirebiter 5 hours ago [-]

"Hints for Computer System Design" by Butler Lampson

https://www.microsoft.com/en-us/research/wp-content/uploads/...

reply

ratsimihah 14 hours ago [-]

Deepmind's first paper on deep reinforcement learning. The beginning of a new era towards AGI : )

Human-level control through deep reinforcement learning
http://www.nature.com/nature/journal/v518/n7540/full/nature1...

reply

dansto 5 hours ago [-]

PCP theorem as explained by Bernard Chazelle , 2001
https://www.cs.princeton.edu/~chazelle/pubs/bourbaki.pdf

Great writing style!

reply

Rickasaurus 12 hours ago [-]

"NP-complete Problems and Physical Reality" by Scott Aaronson. It relates NP-complete problems to examples in nature. Excellent paper and a fun read.

https://arxiv.org/abs/quant-ph/0502072

reply

kwisatzh 20 hours ago [-]

How to share a secret by Adi Shamir. Simple, elegant, short and highly impactful.

reply

> llimllib 20 hours ago [-]
>
> http://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf
>
> reply

Phithagoras 21 hours ago [-]

Not exactly CS, but the Unreasonable Effectiveness of Mathematics in the Natural Sciences is one of my favourites.

reply

johnny_1010 5 hours ago [-]

Communicating Sequential Processes C. A. R. Hoare http://www.usingcsp.com/cspbook.pdf

reply

codelord 9 hours ago [-]

ImageNet Classification with Deep Convolutional Neural Networks https://papers.nips.cc/paper/4824-imagenet-classification-wi... If not evident already, time will tell that this paper brought us to a new era.

reply

vsrinivas 16 hours ago [-]

From the perspective of - 'take a fresh look at something we take for granted' - "A Preliminary Architecture for a Basic Data-Flow Processor" (Dennis & Misunas 1975)

Focusing on the flow of data between operators and greedily executing a linear program is what an out-of-order processor is.

reply

baddox 14 hours ago [-]

Scott Aaronson's "Why Philosophers Should Care About Computational Complexity"

https://www.scottaaronson.com/papers/philos.pdf

reply

cjbprime 8 hours ago [-]

I'll go with an unconventional choice: Michael Kleber's _The Best Card Trick_: http://www.northeastern.edu/seigen/11Magic/Articles/Best%20C...

reply

pradn 21 hours ago [-]

My favorite paper in computer systems is "Memory Resource Management in VMware ESX Server". It identifies a problem and devises several clever solutions to the problem. I love papers that make your go "AHA!".

http://web.eecs.umich.edu/~mosharaf/Readings/ESX-Memory.pdf

reply

nonsince 20 hours ago [-]

Type Systems as Macros

http://www.ccs.neu.edu/home/stchang/popl2017/

It's not world-changing or even particularly novel, but it's such a simple concept explained very well that really changes how you see the typed/dynamic language divide, as well as language design in general.

reply

acoravos 19 hours ago [-]

"The Moral Character of Cryptographic Work" by Phillip Rogaway

Background: http://web.cs.ucdavis.edu/~rogaway/papers/moral.html

Paper: web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf

reply

jhpriestley 20 hours ago [-]

The Scheme papers are great http://library.readscheme.org/page1.html

"On the Translation of Languages from Left to Right", by Knuth, I found much clearer and more illuminating than any of the secondary literature on LR(k) parsing.

reply

twoodfin 20 hours ago [-]

I recall finding "Lambda: The Ultimate GOTO" to be mind-bending.

reply

mrlyc 14 hours ago [-]

My favourite is "Targeting Safety-Related Errors During Software Requirements Analysis" by Robyn Lutz at the Jet Propulsion Laboratory. It's available at https://trs.jpl.nasa.gov/bitstream/handle/2014/35179/93-0749...

The article provides a safety checklist for use during the analysis of software requirements for spacecraft and other safety-critical, embedded systems.

reply

chowells 15 hours ago [-]

The Essence of the Iterator Pattern, by Gibbons and Oliveira.

This paper develops a precise model for internal iteration of a data structure, such that exactly the necessary information is exposed and no more.

It's a fantastic exploration of improving a well-known design space with justified removal of details. I keep its lessons in mind whenever I am facing code that seems to have a lot of incidental complexity.

https://www.cs.ox.ac.uk/jeremy.gibbons/publications/iterator...

reply

efferifick 21 hours ago [-]

Producing Wrong Data without Doing Anything Obviously Wrong.

Immediately useful for anyone measuring compiler transformations performance!

reply

michaelmior 12 hours ago [-]

Link: https://www.eecs.northwestern.edu/~robby/courses/322-2013-sp...

reply

tom_mellior 20 hours ago [-]

Great paper, yes. Immediately useful? More like disheartening, because it doesn't really tell you how to be sure your measurements are OK.

reply

jvilk 19 hours ago [-]

Our lab addressed some of the issues with Stabilizer [0], which "eliminates measurement bias by comprehensively and repeatedly randomizing the placement of functions, stack frames, and heap objects in memory".

[0] http://plasma.cs.umass.edu/emery/stabilizer.html "Stabilizer: Statistically Sound Performance Evaluation" by Charlie Curtsinger and Emery Berger, ASPLOS 2013

reply

jules 20 hours ago [-]

A play on regular expressions: https://sebfisch.github.io/haskell-regexp/regexp-play.pdf

This paper explains a beautiful algorithm for matching regular expressions with a Socratic dialogue.

reply

Jtsummers 21 hours ago [-]

Not a paper, and not strictly CS, but *Mythical Man-Month* by Brooks. It solidified the connection in my mind between systems engineering and software engineering. Other readings since then have extended and changed this understanding, but this is where my approach to software development started to mature.

reply

monocasa 20 hours ago [-]

And I'd add the paper that inspired Brooks, Melvin Conway's "How Do Committees Invent?".

http://www.melconway.com/Home/Committees_Paper.html

reply

sova 18 hours ago [-]

"Collaborative creation of communal hierarchical taxonomies in social tagging systems"

http://ilpubs.stanford.edu:8090/775/1/2006-10.pdf

reply

surement 12 hours ago [-]

The splay trees paper by Sleator and Tarjan (1985)
https://www.cs.cmu.edu/~sleator/papers/self-adjusting.pdf

It's just such a cool result and the paper is very well written. Further, the dynamic optimality conjecture at the end is still an open problem.

reply

jonbaer 13 hours ago [-]

Anything dealing w/ "reversible computing", makes you ask "what-if" ...

http://cqi.inf.usi.ch/qic/80_Toffoli.pdf

https://www.eng.famu.fsu.edu/~mpf/Frank-99-PhD-bookmarked.pd...

reply

coldcode 17 hours ago [-]

Royce 1970 of course: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/wate... wherein he did not introduce Waterfall, but for some reason the negative aspects of his article became the basis for Waterfall. The article for 1970 is surprisingly relevant although archaic in language. It's worth reading to the end. He wrote this describing leading teams in the 1960s do what I assume was actual "rocket" science.

reply

wlesieutre 19 hours ago [-]

"Interactive Indirect Illumination Using Voxel Cone Tracing" by Crassin et al.

As an architectural lighting guy, seeing realtime global illumination look this good in a game engine was fantastic. Parts of the algorithm I can understand, parts go over my head still, but the results are amazing.

A big part of what I do at work is radiosity simulations in AGI32 which is of course more accurate (because it's trying to accurately simulate real world lighting results) but much much slower.

http://research.nvidia.com/publication/interactive-indirect-...

reply

web007 19 hours ago [-]

That link doesn't work for me, even though the same link is presented in a Google search. Also, for some reason I can't find it via search by DOI.

http://research.nvidia.com/publication/interactive-indirect-... has the paper plus presentation available, plus abstract and other links.

reply

wlesieutre 18 hours ago [-]

Thanks! I've replaced it with your working nvidia link.

I got the citeseerx.ist.psu.edu link off google and it worked for me the first time. I'm an alum but not logged into anything, so either it was a bypass via google referrer (not sure why they'd do that, it's not looking for advertising clicks) or some usage limit.

reply

phamilton 12 hours ago [-]

The Dynamo Paper. http://www.allthingsdistributed.com/files/amazon-dynamo-sosp...

One of the best practical "How can this improve our business?" technical papers, and an excellent introduction to reading papers.

reply

505 14 hours ago [-]

I see some of my favourites among other replies. I don't think I see these:

http://aegis.sourceforge.net/auug97.pdf

http://www.cs.virginia.edu/~evans/cs655/readings/steele.pdf

reply

tjr 21 hours ago [-]

*Growing a Language*

reply

andars 19 hours ago [-]

Guy L. Steele.

https://www.cs.virginia.edu/~evans/cs655/readings/steele.pdf

reply

haihaibye 11 hours ago [-]

Why it is Important that Software Projects Fail

http://berglas.org/Articles/ImportantThatSoftwareFails/Impor...

reply

alok-g 21 hours ago [-]

Automated Theorem Proving, David Plaisted

http://onlinelibrary.wiley.com/doi/10.1002/wcs.1269/full

reply

kageneko 16 hours ago [-]

Oh man... I don't know. There's so many.

I'll need to go with

Gilbert, E., & Karahalios, K. (2009, April). Predicting tie strength with social media. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 211-220). ACM.

In grad school, it was the paper that kept on giving. I think I cited it every semester for a paper or project. There's a lot of other papers and books that really inspired me, but this one was magic.

reply

cozzyd 20 hours ago [-]

http://www.scs.stanford.edu/~dm/home/papers/remove.pdf

reply

whataretensors 19 hours ago [-]

The original GAN paper was pretty big for me. https://arxiv.org/pdf/1406.2661.pdf

reply

lukego 21 hours ago [-]

Back To The Future (Squeak). http://ftp.squeak.org/docs/OOPSLA.Squeak.html

reply

0xf8 21 hours ago [-]

The Applications of Probability to Cryptography - Alan M. Turing

https://arxiv.org/abs/1505.04714

reply

Vervious 18 hours ago [-]

Paxos made simple. It's a very beautiful paper.

reply

djhworld 18 hours ago [-]

Not hardcore CS as some of the other papers on here, but I really enjoyed the BigTable paper https://static.googleusercontent.com/media/research.google.c...

reply

donquichotte 21 hours ago [-]

Not CS, but control theory: "Guaranteed Margins for LQG Regulators" by John C. Doyle. The abstract is just three words: "There are none."

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.362...

reply

ddebernardy 16 hours ago [-]

Knuth vs Email:

http://www-cs-faculty.stanford.edu/~knuth/email.html

It's not technically a CS paper, but well worth the (very short) read regardless.

reply

totalZero 16 hours ago [-]

Intelligence without representation, by Rodney Brooks.

http://people.csail.mit.edu/brooks/papers/representation.pdf

reply

Hernanpm 20 hours ago [-]

I sill find this interesting, if you are familiar with Dijkstra Algorithm.

Finding the k Shortest Paths by D. Eppstein https://www.ics.uci.edu/~eppstein/pubs/Epp-SJC-98.pdf

reply

damontal 17 hours ago [-]

the report on the Therac-25. a good warning that bugs can have very real consequences.

http://sunnyday.mit.edu/papers/therac.pdf

reply

zachsnow 20 hours ago [-]

Olin Shivers's work on various control flow analyses, in particular the paper "CFA2: a context-free approach to control-flow analysis", is a really cool static analysis via abstract interpretation. Matt Might had a bunch of papers in a similar vein.

reply

jules 19 hours ago [-]

I wonder if it is possible to express this analysis as a datalog program.

reply

chajath 13 hours ago [-]

Delegation is Inheritance

http://wiki.c2.com/?DelegationIsInheritance

reply

hatred 17 hours ago [-]

The Byzantine Generals Problem by Lamport et al is a must read for anyone interested in distributed systems.

Some of the others that have already been mentioned on this thread:

- Time, Clocks, and the Ordering of Events in a Distributed System

- Paxos Made Simple

reply

di4na 18 hours ago [-]

"Programming with Abstract Data Types", B Liskov and S Zilles

reply

wwarner 20 hours ago [-]

Probabilistic Topic Models http://www.cs.columbia.edu/~blei/papers/Blei2012.pdf

reply

morphle 19 hours ago [-]

Scalability of Collaborative Environments https://sci-hub.ac/10.1109/C5.2006.32#

reply

notaharvardmba 20 hours ago [-]

Andrew Tridgell's PhD Thesis: https://www.samba.org/~tridge/phd_thesis.pdf

Which documents the invention of rsync, it's a good read.

reply

gcp 19 hours ago [-]

His KnightCap papers are also cool, .e.g http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124...

AlphaGo learning from self-play? Tridgell did it in 1998.

reply

nayuki 20 hours ago [-]

"Bitcoin: A Peer-to-Peer Electronic Cash System" https://bitcoin.org/bitcoin.pdf

reply

AnimalMuppet 19 hours ago [-]

Why Pascal Is Not My Favorite Programming Language, by Brian Kernighan http://www.cs.virginia.edu/~cs655/readings/bwk-on-pascal.htm...

No Silver Bullet, by Fred Brooks http://worrydream.com/refs/Brooks-NoSilverBullet.pdf

The original STL documentation https://www.sgi.com/tech/stl/table_of_contents.html

reply

gnaritas 20 hours ago [-]

Not a paper, but one of my favorite talks

https://www.youtube.com/watch?v=_ahvzDzKdB0&t=642s

Growing a Language by Guy Steele (co-inventory of Scheme). Brilliant speech about how to grow languages and why it's necessary. Languages that can be grown by the programmer, like Lisp or Smalltalk are better than languages that fixed like most others, this is why.

reply

probinso 13 hours ago [-]

Relevance Vector Machines by Tipping

or

Homomorphic Encryption over the Integers

reply

megamindbrian 21 hours ago [-]

My favorite topic was from an advanced user interfaces class. Describe 3 example of a bad user experience where the input in to the system does not give you the expected output. My poor example was a Kleenex box, I try to pull on one Kleenex and it tears or two come out at a time.

reply

throwaway1e100 21 hours ago [-]

Real programmers don't use Pascal The rise of worse is better

reply

kruhft 12 hours ago [-]

On Formally Undecidable Propositions... by Kurt Godel.

One might argue this is not CS, but it's something everyone should read and understand.

reply

kendallpark 16 hours ago [-]

I'm surprised no one has mentioned "The Cathedral and the Bazaar" yet. Admittedly is more essay, less paper.

https://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar

reply