ANALIZA CATEGORICĂ A ALGORITMILOR DE SORTARE

Petru CERVAC

Faculty of Computers, Informatics andMicroelectronics, Technical University of Moldova, Chișinău, Moldova

Autorul corespondent: Petru CERVAC, e-mail cervac.petru@doctorat.utm.md

Coordonator științific: Viorica SUDACEVSCHI, dr. conf. Univ., UTM

Rezumat. *Acest articol explorează posibilitatea utilizării teoriei categoriilor pentru a analiza complexitatea algoritmilor de sortare. Sunt analizați diferiți algoritmi de sortare, cum ar fi bubble sort, insertion sort, selection sort, merge sort, quicksort și heapsort, din punct de vedere al complexității lor în timp și spațiu. Apoi sunt descrise conceptele din teoria categoriilor, inclusiv functorialitatea, transformările naturale și adjuncțiile, și se explică cum acestea pot fi utilizate pentru analiza algoritmiilor de sortare într-un mod sistematic și unificat. Studiul propus în lucrare demonstrează potențialul teoriei categoriilor în analiza comportamentului algoritmilor de sortare și oferirea unei abordări pentru compararea și contrastarea diferitor algoritmi. Sunt discutate limitările și provocările utilizării teoriei categoriilor în practică și se propun căi pentru cercetarea viitoare. Lucrarea contribuie la dialogul continuu despre rolul teoriei categoriilor în informatică și complexitatea computațională.*

Cuvinte cheie*: algoritmi de sortare, teoria categoriilor, complexitate, eficiență.*

Introducere

Algoritmii de sortare sunt esențiali în informatică, servind drept fundament pentru mulți alți algoritmi. Aceștia se numără printre cei mai frecvent utilizați algoritmi în calculatoare. În plus, algoritmii de sortare sunt fundamentali pentru dezvoltarea algoritmilor mai avansați, cum ar fi căutarea binară, algoritmul lui Dijkstra și algoritmul lui Prim. În plus, algoritmii de sortare sunt frecvent utilizați în mediul academic pentru a-i introduce pe studenți în noțiunile de bază ale algoritmilor. Acest lucru se datorează faptului că sortarea poate fi realizată printr-un program de calculator relativ simplu, dar nebanal, implementat într-o varietate de moduri.

Sortarea este un proces de aranjare a datelor într-o anumită ordine, bazată pe relația dintre elemente. Aceasta poate implica sortarea datelor într-o ordine crescătoare sau descrescătoare. Deși adesea asociată cu calculatoarele, sortarea este o sarcină comună în multe scenarii din lumea reală. De exemplu, bibliotecile organizează cărțile după numele autorului și categorie, dicționarele sortează cuvintele în ordine alfabetică, facilitățile poștale sortează coletele pentru livrare, iar producătorii agricoli aranjează produse precum merele sau morcovii după mărime.

Sortarea a fost una dintre problemele pe care tehnologia a ajutat să le optimizeze. De fapt, prima mașină de sortare a fost dezvoltată în Statele Unite în secolului al XIX-lea pentru a ajuta la recensământul populației. Ulterior, John von Neumann a dezvoltat primul procedeu de sortare pentru un calculator, mai exact algoritmul de sortare prin interclasare pentru computerul EDVAC [1].

Sortarea este un subiect de mare interes printre cercetători. În ciuda aparentei sale simplități, sortarea ascunde o mulțime de mistere și întrebări fără răspuns. De exemplu, numărul minim de comparații necesare pentru a sorta un tablou de peste 47 de elemente, existența unui algoritm care utilizează mai puține comparații decât algoritmul Ford-Johnson, proiectarea unui algoritm mai eficient pentru rețele de sortare decât sortarea prin interclasare bitonică,schema optimă de divizare pentru algoritmul Shell - sunt doar unele întrebări care rămân fără răspuns.

Cea mai cuprinzătooare resursă despre sortare rămâne cartea "Arta programării calculatoarelor" de Knuth [1]. A fost lansată inițial în 1967, cea mai recentă fiind a doua ediție, lansată în 1998. Cartea descrie majoritatea familiilor de algoritmi de sortare, proprietățile matematice ale sortării, precum și analiza optimă a sortării. Cartea menționează o serie de întrebări deschise în

domeniu care au servit sursă de inspirație pentru cercetarea în secolul al XXI-lea. Un subiect pe care Knuth nu l-a inclus în cartea sa este analiza optimă a sortării bazate pe alte metode decât comparația, ceea ce este regretabil.

Sortarea este o operație crucială în informatică, iar căutarea celei mai optime metode a fost întotdeauna de mare interes. Minimizarea resurselor utilizate de un algoritm de sortare este un obiectiv important. Deși în anii 1960 au fost dezvoltate tehnici pentru analizarea complexității algoritmilor , puține activități în acest domeniu au avut loc de atunci. Autorul consideră că teoria categoriilor ar putea fi un domeniu aplicat pentru analizarea sortării și a algoritmilor în general. Deși există puține lucrări care explorează această idee, lucrările recente ale lui Hinze [2, 3] referitor la proprietățile categorice ale familiilor de sortare prin inserție și selecție și ale lui Basu [4] asupra complexității categorice, arată promițător. Lipsa cercetării în domeniul analizei algoritmice prin prisma teoriei categoriilor este surprinzătoare, dat fiind rolul său tradițional ca sursă de inspirație pentru noi dezvoltări în proiectarea limbajelor de programare, în special în limbajele de programare funcționale. Teoria categoriilor este o unealtă potențială pentru analiza performanței algoritmului și ar trebui să fie explorată în continuare în acest context.

Scopul acestui articol este studierea perspectivelor utilizării teoriei categoriilor pentru analiza complexității algoritmilor. În Capitolul 1 este discutată starea actuală a cercetării în domeniul sortării. În Capitolul 2 sunt prezentați algoritmii de sortare.Capitolul 3 este dedicat teoriei categoriilor. În Capitolul 4 este discutată complexitatea algoritmilor.

Starea actuală a cercetării

În prezent, cercetarea în domeniul de sortare se concentrează pe mai multe direcții, inclusiv dezvoltarea de proceduri noi pentru arhitecturile paralele precum GPU, procesoare multithreaded și FPGA. Începând cu mijlocul anilor 2000, legea lui Moore a fost susținută prin îmbunătățirea numărului de nuclee pe un singur procesor, dar programarea multithreaded este notoriu dificilă. În timp ce algoritmii cu un singur fir sunt mai ușor de înțeles și de implementat, crearea unei versiuni multithreaded a algoritmilor utilizați frecvent este necesară pentru a utiliza pe deplin potențialul computerelor moderne.

Dezvoltarea algoritmilor de sortare pentru arhitecturi paralele, cum ar fi GPU-urile, este deosebit de interesantă, deoarece acestea au mii de nuclee și pot fi destul de eficiente în rezolvarea anumitor tipuri de probleme. Mai multe lucrări au explorat acest domeniu [5, 6].

Sortarea pe FPGA este, de asemenea, destul de interesantă. FPGA sunt circuite complexe care pot fi reprogramate. FPGA pot fi folosite pentru a construi cipuri personalizate care rezolvă o problemă specifică foarte bine. Acestea sunt de obicei implementate în aplicații în care viteza de procesare a datelor de intrare este critică, cum ar fi avionica, automobilele și tranzacțiile. În multe astfel de aplicații, sortarea este una dintre etapele principale. Accelerarea procesului de sortare folosind FPGA ar putea fi o soluție viabilă. Există eforturi de portare a algoritmilor de sortare existenți, cum ar fi bubble sort pe FPGA [7]. Există, de asemenea, eforturi în dezvoltarea de algoritmi de sortare special concepuți pentru a fi implementați pe FPGA. Rețelele de sortare sunt deosebit de potrivite pentru FPGA [8]. Numărul minim de comparatoare necesare pentru a sorta mai mult de 16 elemente este o întrebare actuală [9], precum și dezvoltarea unui algoritm care oferă o rețea de sortare mai optimă decât bitonic mergesort [1].

Big data este un alt subiect dificil, când vine vorba de sortare. Cantitatea de date crește exponențial, iar mai multe aplicații utile sunt dezvoltate pentru a utiliza big data. Cantitatea de date limitează abordările tradiționale de procesare a datelor. Sortarea este un pas cheie în multe dintre procesările de big data, așadar este important să dezvoltăm algoritmi de sortare eficienți pentru astfel de scenarii [10].

Proprietățile teoretice ale sortării sunt, de asemenea, analizate, chiar dacă nu la fel de mult, ca în deceniile anterioare. În 2022 a fost calculat numărul minim de comparații necesare pentru a sorta 16, 17 și 18 elemente [11]. Alte cercetări în acest domeniu sunt analizate în [12] și [13]. De asemenea, sunt dezvoltate noi algoritmi de sortare de uz general [14, 15].

Un domeniu care merită mai multă atenție este analiza algoritmilor de sortare non-comparație. Cercetarea activă în această direcție a început în anii 1890 cu lucrarea lui Kirkpatrick și David [16], care a arătat limitele superioare pentru sortarea numerelor întregi pe modelul RAM nelimitat. Întrebările deschise și limitările din această lucrare au servit ca sursă de inspirație pentru cercetarea efectuată în anii 1990 și 2000 [17, 18] (Han & Thorup, 2002)). Cel mai recent articol găsit de autor este din anul 2020, în care se introduce un nou algoritm de sortare a numerelor întregi numit *RadixInsert*, utilizat pentru sortarea numerelor cu virgulă mobilă [15]. Starea actuală a cercetării algoritmilor de sortare non-comparație este importantă, deoarece acești algoritmi sunt mai eficienți decât cei bazati pe comparație, datorită utilizării tuturor proprietăților datelor de intrare și ale mașinii subiacente. Studierea algoritmilor de sortare non-comparație este utilă și pentru înțelegerea tuturor algoritmilor în general, ca un studiu de caz privind modul în care structura datelor de intrare influențează performanța algoritmului.

Clasificarea algoritmilor de sortare

Sortare internă vs externă. Algoritmii de sortare pot fi clasificați ca fiind interni sau externi, în funcție de faptul dacă înregistrările sunt păstrate integral în memoria calculatorului (intern) sau dacă se utilizează stocarea externă și doar o cantitate limitată de înregistrări este procesată într-un moment dat (extern) [16]. Algoritmii de sortare interni sunt mai des utilizați în zilele noastre, deoarece calculatoarele au o cantitate mare de memorie și nu au o problemă în stocarea cantității de date cu care interacționează un utilizator tipic. Sortarea externă este un subiect de cercetare aparte.

Sortarea bazată pe comparații vs sortarea nebazată pe comparații. Dacă un algoritm de sortare utilizează o operație de comparare, acesta este denumit sortare bazată pe comparații, în caz contrar, este un algoritm nebazat pe comparații. Cei mai cunoscuți algoritmi de sortare (ex. BubbleSort, MergeSort, SelectionSort, QuickSort etc.) sunt algoritmi de sortare bazată pe comparații. Algoritmii de sortare nebazati pe comparații (ex. BucketSort, RadixSort) utilizează proprietățile matematice ale înregistrărilor întregi pentru a elimina complet operația de comparare.

Sortare stabilă vs instabilă. O sortare este numită *stabilă* dacă înregistrările cu chei egale își păstrează ordinea originală. În caz contrar, algoritmul de sortare este considerat *instabil* [1].

Conservator vs non-conservator. Un algoritm de sortare este considerat *conservator* dacă toate rezultatele intermediare încap în memoria computerului, în caz contrar, algoritmul este *non-conservator* [16].

Sortarea prin inserție. Ideea din spatele acestei metode este de a lua un element la fiecare pas și de a-l insera într-un tablou deja sortat. Implementarea directă a algoritmului are o complexitate de timp de $O(n^2)$. Aceasta poate fi redusă prin utilizarea inserției binare și a inserției cu două direcții. Această tehnică este cea mai potrivită pentru tablouri implementate ca liste înlănțuite, deoarece acestea minimizează timpul necesar pentru inserarea elementului la loc și se reduce la înlocuirea valorii unui pointer cu unul diferit. Unul dintre cei mai cunoscuți algoritmi din această categorie este shellsort, dezvoltat de Donald L. Shell, în 1959. Acesta funcționează prin subdivizarea tabloului original în grupuri de subtablouri și inserarea elementelor în fiecare subtablou. Cu toate acestea, încă există întrebări de cercetare deschise despre acesta, cum ar fi care este cea mai bună secvență de creștere pentru valori mari ale lui N. Shellsort este locul de naștere al rețelelor de sortare, o poveste fascinantă în sine. Această tehnică este cea mai potrivită pentru structurile de date care minimizează timpul de inserție, cum ar fi listele înlănțuite. O aranjare structurată ca un arbore este, de asemenea, potrivită pentru aceste tipuri de algoritmi.

Sortarea prin interschimbare. Ideea din spatele sortării prin interschimbare este de a interschimba sistematic o pereche de elemente care sunt în ordine greșită până când nu mai există astfel de elemente. Exemple de sortări din această familie de algoritmi sunt bubble sort și quick sort.

Sortare pentru selectare. Ideea din spatele sortării prin selecție este de a selecta continuu valoarea minimă dintr-un array și plasarea ei la începutul părții sortate a array-ului. Din această familie de sortare fac parte sortarea prin selecție directă și sortarea prin intercalare (heap sort).

Sortare prin combinare. Merge sort este un algoritm de sortare popular care utilizează o strategie ***divide and conquer*** pentru a sorta o listă de elemente. Algoritmul se bazează pe împărțirea recursivă a listei în subliste mai mici, sortarea fiecărei subliste și apoi combinarea sublistelor sortate împreună pentru a produce lista finală sortată.

Sortare prin distribuție. Sortarea prin distribuție, cunoscută și sub denumirea de sortare prin găleți sau sortare radix, este un algoritm de sortare care funcționează prin gruparea elementelor unei liste în găleți, bazate pe valorile lor, apoi sortând elementele în fiecare găleată individual și, în final, combinând gălețile sortate pentru a obține lista sortată finală.

Teoria categoriilor pentru analiza complexității

Teoria categoriilor este o ramură a matematicii care studiază o structură particulară numită categorie. O categorie este o colecție de obiecte care sunt legate între ele prin morfisme. Există câteva reguli la care morfismele și obiectele trebuie să se conformeze:

- **Composabilitate**: Dacă $f: a \rightarrow b,\ g: b \rightarrow c$, atunci $g \circ f: a \rightarrow c$
- **Asociativitate**: Dacă $f: a \rightarrow b, g: b \rightarrow c$ și $h: c \rightarrow d$ atunci $h \circ (g \circ f) = (h \circ g) \circ f$
- **Identitate**: pentru fiecare obiect x, există un morfism $id_x: x \rightarrow x$ numit morfismul identității pentru x, astfel încât fiecare morfism $f: a \rightarrow x$ să satisfacă $id_x \circ f = f$ și fiecare morfism $g: x \rightarrow b$ să satisfacă $g \circ id_x = g$.

Deși conceptul de categorie poate fi considerat rudimentar, aplicabilitatea sa în formalizarea diverselor structuri matematice este extensivă. Teoria categoriilor ajută la descrierea modului în care conceptele sunt legate între ele, fiind vorba despre relațiile dintre specii de animale sau relațiile dintre structuri matematice abstracte, cum ar fi grupuri și spații topologice. Aplicarea conceptelor teoriei categoriilor a devenit o sursă semnificativă pentru stimularea inovației în domeniul dezvoltării de limbaje de programare, în special a limbajelor de programare funcționale. Haskell, un limbaj de programare funcțional prominent, este un exemplu elocvent despre modul în care ideile din teoria categoriilor au fost instrumentale în modelarea design-ului și implementării limbajului.

Teoria categoriilor nu este utilizată atât de mult în domeniul analizei computaționale. Hinze și James au utilizat teoria categoriilor pentru a analiza tehnici de sortare în programarea funcțională prin intermediul bialgebrelor și legilor distributive [3] (Hinze, James, Harper, Wu, & Magalhães, 2012). Aceștea arată dualitatea abordărilor de îmbinare și de selecție pentru sortare. Ei își continuă analiza asupra dualității metodelor de sortare într-un articol ulterior [2]. Basu introduce domeniul complexității categorice și al calculului categoric. El îl utilizează pentru a deriva complexitatea mai multor structuri matematice [4].

Autorul crede că teoria categoriilor are potențial în ceea ce privește analiza algoritmilor. În cea mai rudimentară formă, un algoritm poate fi înțeles ca o transformare între intrare și ieșire. Intrarea și ieșirea au anumite structuri asociate lor. Înțelegerea structurii intrării și ieșirii va aduce evident o mai bună înțelegere a transformării dintre ele. Un algoritm de sortare poate fi perceput ca o transformare între categoria de liste nesortate și categoria de liste sortate.

Complexitate și eficiență

Algoritmii de sortare sunt caracterizați prin complexitatea lor în timp și spațiu, care sunt măsurile fundamentale ale eficienței lor. Complexitatea în timp se referă la cantitatea de timp necesară pentru ca un algoritm să ruleze în funcție de dimensiunea datelor de intrare. Complexitatea în spațiu, pe de altă parte, se referă la cantitatea de memorie necesară pentru ca un algoritm să ruleze în funcție de dimensiunea datelor de intrare. Aceste complexități sunt adesea exprimate utilizând notația big O, care oferă o limită superioară a ratei de creștere a utilizării resurselor algoritmului în funcție de datele de intrare.

Notația big O este definită ca fiind mulțimea tuturor funcțiilor care nu cresc mai rapid decât o anumită funcție dată, până la un factor constant. Mai formal, $O(f(n))$ denotă mulțimea tuturor funcțiilor $g(n)$ astfel încât există constante pozitive C și n_0 astfel încât $|g(n)| \leq Cf(n)$, pentru toate $n \geq n_0$. Notația big Omega, denumită $\Omega(f(n))$, furnizează o limită inferioară a ratei de creștere a

utilizării resurselor algoritmului în funcție de datele de intrare. Conceptul de notație big O și big Omega pentru analizarea performanței algoritmilor a fost popularizat de Donald Knuth.

Analiza performanței algoritmilor este o problemă complexă și subiectivă la mulți factori, inclusiv arhitectura calculatorului, calitatea implementării, limbajul de programare și altele. Există două abordări principale pentru analiza algoritmilor: teoretică și empirică.

În abordarea empirică, un algoritm este executat pe un hardware specific cu diferite dimensiuni ale datelor de intrare pentru a observa cantitatea de resurse pe care o consumă. Această abordare este utilă pentru a determina comportamentul unui algoritm pe hardware-ul real și poate ajuta la răspunsul unor întrebări care sunt dificil de rezolvat teoretic. Cu toate acestea, această abordare are și dezavantaje, inclusiv riscul de a afecta accidental performanța prin arhitectură, imposibilitatea de a răspunde la toate întrebările și dificultatea de a valida rezultatele pentru toate scenariile.

Analiza teoretică a algoritmilor implică utilizarea unui model computațional, cum ar fi o mașină Turing sau lambda calculus, pentru analiza algoritmilor. Avantajul acestei abordări este că permite demonstrația matematică a rezultatelor, iar operațiile definite în model pot fi folosite pentru a deduce numărul de resurse utilizate în funcție de dimensiunea datelor de intrare. Cu toate acestea, rezultatele analizei teoretice pot diverge de performanța reală, deoarece modelele de calcul sunt adesea mai simple decât arhitecturile realizate, iar operațiile definite în modele pot fi departe de implementările din lumea reală.

Analiza sortării bazate pe comparații. Numărul de comparații utilizat de un algoritm este un factor important pentru determinarea eficienței sale, deoarece comparațiile sunt operații costisitoare, în mod tradițional. Prin minimizarea numărului de comparații necesare, putem îmbunătăți eficiența algoritmului. Se știe că un algoritm de sortare bazat pe comparații are o complexitate a timpului de cel mult $O(n \log n)$ datorită cantității de informații pe care o oferă fiecare comparație. Cu toate acestea, nu există un algoritm cunoscut care să atingă această limită teoretică pentru toate dimensiunile datelor de intrare. Determinarea numărului exact de comparații necesare pentru a sorta un număr dat de intrări este o problemă computațională dificilă. Merge Insertion sort, cunoscut și sub numele de algoritmul de sortare Ford-Johnson, este în prezent cel mai eficient algoritm cunoscut pentru sortarea listelor cu mai puțin de 47 de elemente [19]. Acesta atinge limita inferioară teoretică a informației pentru $N \leq 22$, iar numărul exact de comparații necesare a fost demonstrat pentru matrice cu mai puțin de 22 de elemente [1, 11, 20, 21]. Cu toate acestea, algoritmul nu este cel mai eficient pentru toate valorile de N [22].

Analiza sortării non-bazate pe comparații. Algoritmii de sortare non-bazați pe comparații au, de obicei, o performanță asimptotică mai bună decât cei care se bazează pe comparații, deoarece nu sunt restricții la utilizarea doar a operatorului de comparație. În schimb, aceștia pot folosi mai multe informații disponibile în cheie. Algoritmii de sortare non-bazați pe comparații pot gestiona structuri de chei mai complexe, cum ar fi numerele întregi, ceea ce permite utilizarea unei game mai largi de operații. Este important să se aleagă modelul corect de calcul atunci când se evaluează performanța algoritmilor de sortare. Limita inferioară pentru sortarea bazată pe numere întregi a fost determinată a fi $O(n)$, deși pentru constante prea mari pentru a fi practice [16]. În timp ce algoritmii teoretici de sortare a numerelor întregi au o performanță asimptotică bună, aceștia nu sunt practici pentru a fi implementați pe hardware real. Cercetarea în domeniul sortării non-bazați pe comparații a început în anii 1980, cu cei mai eficienți algoritmi de sortare non-bazați pe comparații, și anume counting sort, radix sort, bucket sort, RadixInsert și Signature sort [18]. Acești algoritmi sunt, de asemenea, cunoscuți sub numele de algoritmi de sortare bazate pe numere întregi, care au realizat limite teoretice inferioare mari. Cercetarea curentă se concentrează pe furnizarea de implementări paralele ale algoritmilor de sortare a numerelor întregi, în special pe GPU. În prezent, eforturile sunt îndreptate spre cercetarea algoritmilor de sortare a numerelor întregi care pot fi implementați pe o arhitectură paralelă. Algoritmii de sortare conservativă, care utilizează doar comparații și sunt concepuți pentru o gamă fixă de chei, au două idei importante pentru algoritmii de sortare a numerelor întregi: reducerea gamei și sortarea împachetată. O implementare interesantă a sortării întregi este AC^0, care poate fi făcută în timp $O(n)$ și cu cerințe de spațiu $O(1)$ [23].

Concluzii

În concluzie, analiza complexității algoritmilor este crucială în găsirea celei mai eficiente modalități de a realiza sarcina propusă. Teoria categoriilor poate furniza un framework util pentru analiza algoritmilor, iar algoritmii de sortare, în special, sunt un punct de plecare excelent datorită argumentării matematice solide. Combinând teoria categoriilor cu algoritmii de sortare, putem explora întrebări interesante precum definirea de vectori și vectori sortați în contextul categoric, precum și exprimarea proprietăților subiacente ale datelor. Aceste întrebări reprezintă domenii importante pentru cercetare care ar putea duce la noi perspective în analiza algoritmilot. În general, intersecția teoriei categoriilor și a algoritmilor de sortare prezintă o direcție promițătoare pentru avansarea înțelegerii noastre a algoritmilor și a complexității lor.

Bibliografie

1. D. E. KNUTH, The Art of Computer Programming: Volume 3, vol. 3, Addison-Wesley Pub. Co., 1997.
2. R. HINZE, J. P. MAGALHÃES ȘI N. WU, „A Duality of Sorts,” în Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, p. 151–167.
3. R. HINZE, D. JAMES, T. HARPER, N. WU și J. MAGALHÃES, „Sorting with Bialgebras and Distributive Laws,” 2012.
4. S. BASU și M. U. ISIK, „Categorical Complexity,” Forum of Mathematics, Sigma 8 (2020) e34, vol. 8, 2016.
5. S. ABDEL-HAFEEZ, A. GORDON-ROSS și S. ABUBAKER, „A comparison-free sorting algorithm on CPUs and GPUs,” The Journal of Supercomputing, vol. 74, p. 6369–6400, 1 November 2018.
6. N. F. A. S. GHRERA, „Performance Evaluation of Parallel Count Sort using GPU Computing with CUDA,” Indian Journal of Science and Technology, vol. 9, p. 1–12, 14 April 2016.
7. N. LEISCHNER, V. OSIPOV și P. SANDERS, „GPU sample sort,” în 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 2010.
8. H. PETERS, O. SCHULZ-HILDEBRANDT și N. LUTTENBERGER, „Fast In-Place Sorting with CUDA Based on Bitonic Sort,” în Parallel Processing and Applied Mathematics, Berlin, 2010.
9. P. PAPAPHILIPPOU, W. LUK și C. BROOKS, „FLiMS: A Fast Lightweight 2-Way Merger for Sorting,” IEEE Transactions on Computers, vol. 71, p. 3215–3226, December 2022.
10. T. EHLERS, "Merging almost sorted sequences yields a 24-sorter," Information Processing Letters, vol. 118, p. 17–20, February 2017.
11. D. BUNDALA and J. ZÁVODNÝ, "Optimal Sorting Networks," in Language and Automata Theory and Applications, Cham, 2014.
12. SHATNAWI, Y. ALZAHOURI, M. A. SHEHAB, Y. JARARWEH și M. AL-AYYOUB, „Toward a new approach for sorting extremely large data files in the big data era,” Cluster Computing, vol. 22, p. 819–828, 1 September 2019.
13. F. STOBER și A. WEIß, „Lower Bounds for Sorting 16, 17, and 18 Elements,” 2022.
14. K. IWAMA și J. TERUYAMA, „Improved average complexity for comparison-based sorting,” Theoretical Computer Science, vol. 807, p. 201–219, 6 February 2020.
15. S. EHARA, K. IWAMA și J. TERUYAMA, „Small Complexity Gaps for Comparison-Based Sorting,” H. Böckenhauer, D. Komm și W. Unger, Ed., Cham, Springer International Publishing, 2018, p. 280–296.
16. S. P. Y. FUNG, „Is this the simplest (and most surprising) sorting algorithm ever?,” October 2021.
17. MAUS, „RadixInsert, a much faster stable algorithm for sorting floating-point numbers,” 2020.
18. D. KIRKPATRICK and S. REISCH, "Upper bounds for sorting integers on random access machines," Theoretical Computer Science, vol. 28, p. 263–276, January 1983.

19. Y. HAN și M. THORUP, „Integer sorting in O(n/spl radic/(log log n)) expected time and linear space,” în The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., 2002.
20. M. THORUP, "Randomized Sorting in O(nloglogn) Time and Linear Space Using Addition, Shift, and Bit-wise Boolean Operations," Journal of Algorithms, vol. 42, p. 205–230, February 2002.
21. ANDERSSON, T. HAGERUP, S. NILSSON and R. RAMAN, "Sorting in Linear Time?," Journal of Computer and System Sciences, vol. 57, p. 74–93, August 1998.
22. L. R. FORD și S. M. JOHNSON, „A Tournament Problem,” The American Mathematical Monthly, vol. 66, p. 387–389, May 1959.
23. M. PECZARSKI, "New Results in Minimum-Comparison Sorting," Algorithmica, vol. 40, p. 133–145, October 2004.
24. M. PECZARSKI, "The Ford–Johnson algorithm still unbeaten for less than 47 elements," Information Processing Letters, vol. 101, p. 126–128, February 2007.
25. G. K. MANACHER, „The Ford-Johnson Sorting Algorithm Is Not Optimal,” Journal of the ACM, vol. 26, p. 441–456, July 1979.
26. G. FRANCESCHINI, S. MUTHUKRISHNAN ȘI M. PATRASCU, „Radix Sorting With No Extra Space,” 2007.