A Metastudy of Algorithm Lower Bounds

by

Emily Liu

B.S. in Computer Science and Engineering, Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Electrical Engineering and Computer Science
July 9, 2021

Certified by .....
Neil Thompson
Research Scientist
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis

A Metastudy of Algorithm Lower Bounds

by

Emily Liu

Submitted to the Department of Electrical Engineering and Computer Science
on July 9, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Algorithms are essential to the field of computer science, and algorithm designers are always searching for the mathematically optimal algorithms. Sherry and Thompson found that improvements to algorithm upper bounds have been steadily decreasing since the 1970s. In this work we aim to discover whether this could be because researchers have already found the optimal versions of many algorithms. In order to get a better sense of the picture, we compiled lower bounds on the algorithm families studied by Sherry and Thompson. We find that, while a few problems still have large gaps between upper and lower bounds where improvement is possible, over three-quarters of these problems are already very close to being optimal! The "slowing progress" may in fact prove to be a triumph in disguise, as it is an indicator that many problems have achieved optimal solutions.

Thesis Supervisor: Neil Thompson
Title: Research Scientist

Acknowledgments

I feel so lucky to have had such excellent support throughout the process of completing this thesis. I could not have asked for better people in my life during the past year, especially during a pandemic and through a personal emergency.

First and foremost, I would like to thank my thesis advisor, Dr. Neil Thompson. I am grateful that he gave me the opportunity to work with him, on such an interesting and exciting project. Neil's feedback and direction related to the project have been essential, and his consistent support, kindness, and encouragement have been invaluable.

I would also like to thank William Kuszmaul for his mentorship. He provided careful guidance and excellent advice, without which I would certainly have been lost. From walking me through the initial research process to meticulously reading over every line of my manuscript, Bill has always been there to answer any questions and provide feedback.

I am grateful to Yash Sherry for his great help in making this project come together. Yash generously provided resources and worked tirelessly with me to make this project the best it could be.

To my friends, thank you for making my MIT experience so memorable. A special thanks goes to FEHJM House for being a highlight of the past year.

To my family, thank you for supporting and encouraging me always, and for having faith in me. I would not be who I am today without you.

Contributions

This thesis is based on joint work with Dr. Neil Thompson, William Kuszmaul, and Yash Sherry. Neil conceived of and provided direction for the project as a whole, Bill helped shape it with his insights and suggestions for analysis, Yash brought his expertise and experience from his previous work with Neil, and I performed the data collection and analysis. I wrote the thesis, with Bill and Neil reviewing. We are planning to submit a co-authored version of this paper for publication.

Funding

This work was supported in part by the Tides Foundation (1903-57432), NSF grant 2041897, an NSF GRFP fellowship, and a Fannie and John Hertz Fellowship.

Research was also partially sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Contents

List of Figures

Chapter 1

Introduction

An algorithm is an unambiguous sequence of instructions to solve a clearly defined problem. Algorithms are the step-by-step procedures that computers follow, and they are what allow computers to execute so many tasks so cleverly, in turn allowing researchers to approach problems in many different subject areas. The field of computer science has improved by leaps and bounds since it seriously began in the 1940s, and much of the progress has been in or due to algorithms. In fact, for many problems, algorithmic improvements have outpaced hardware improvements [48].

Recent work by Sherry and Thompson has aimed to quantify the level and rate of progress in algorithms research. They compiled a list of the most commonly taught algorithms across several computer science disciplines and researched the history of each one. In order to better track the history of related algorithms, they grouped together algorithms that all solve the same problem into *algorithm families*. Sherry and Thompson's results suggest that progress in algorithms has slowed; in the decades following the 1970s, the number of algorithm families that have been seeing improvements has been decreasing. The natural followup question is, of course—why?

This study explores one possible answer, which is that perhaps the field is approaching optimality. While algorithms have continued to grow better and faster over the years, there comes a point where they reach the limit of what is possible and where they cannot improve any further. This brings us to the concept of lower bounds: A lower bound is a limit that even the best possible algorithm cannot im-

prove beyond. Could it be that many algorithm families have simply reached their lower bounds, and can no longer be made any better? If so, the slowing progress could simply be a side effect of a triumph in disguise.

Our results indicate that this could indeed be the case. More than three-quarters of the analyzed algorithms are asymptotically optimal or close to it—in other words, they are nearly the best and fastest they possibly can be, especially for large problem sizes. It remains to be seen how this will continue to affect the future of algorithm progress.

1.1 Upper and Lower Bounds

Typically, when we think about how "good" an algorithm is, we think about how fast it is, that is, how much time (or how many steps) it takes to complete. One common measure of speed is asymptotic time complexity, which is an upper bound on the amount of time an algorithm will require as the input size grows. This is how we usually quantify "improvement"; when a researcher invents an algorithm with smaller asymptotic time complexity, we say that the algorithm has been improved.

But how much can we continue to improve? In order to answer this question, we need to know some provable lower bound on the time complexity. A lower bound for an algorithm family is a quantity that is provably less than or equal to the best possible running time of any algorithm in that family. We cannot improve past the lower bound, so if the upper bound and lower bound time complexities for a particular problem are the same, then that problem has its optimal time complexity. If they are not the same, then there is the potential for improvement; when a researcher proves a larger lower bound, we say that the lower bound improves.

For example, in the case of sorting n real numbers, one possible lower bound on the number of steps is n, since every single number needs to be put in its place. Here, and in all such problems, n is the size of the input. We then say that sorting has a lower bound on time complexity of $\Omega(n)$. However, this is not the optimal lower bound.

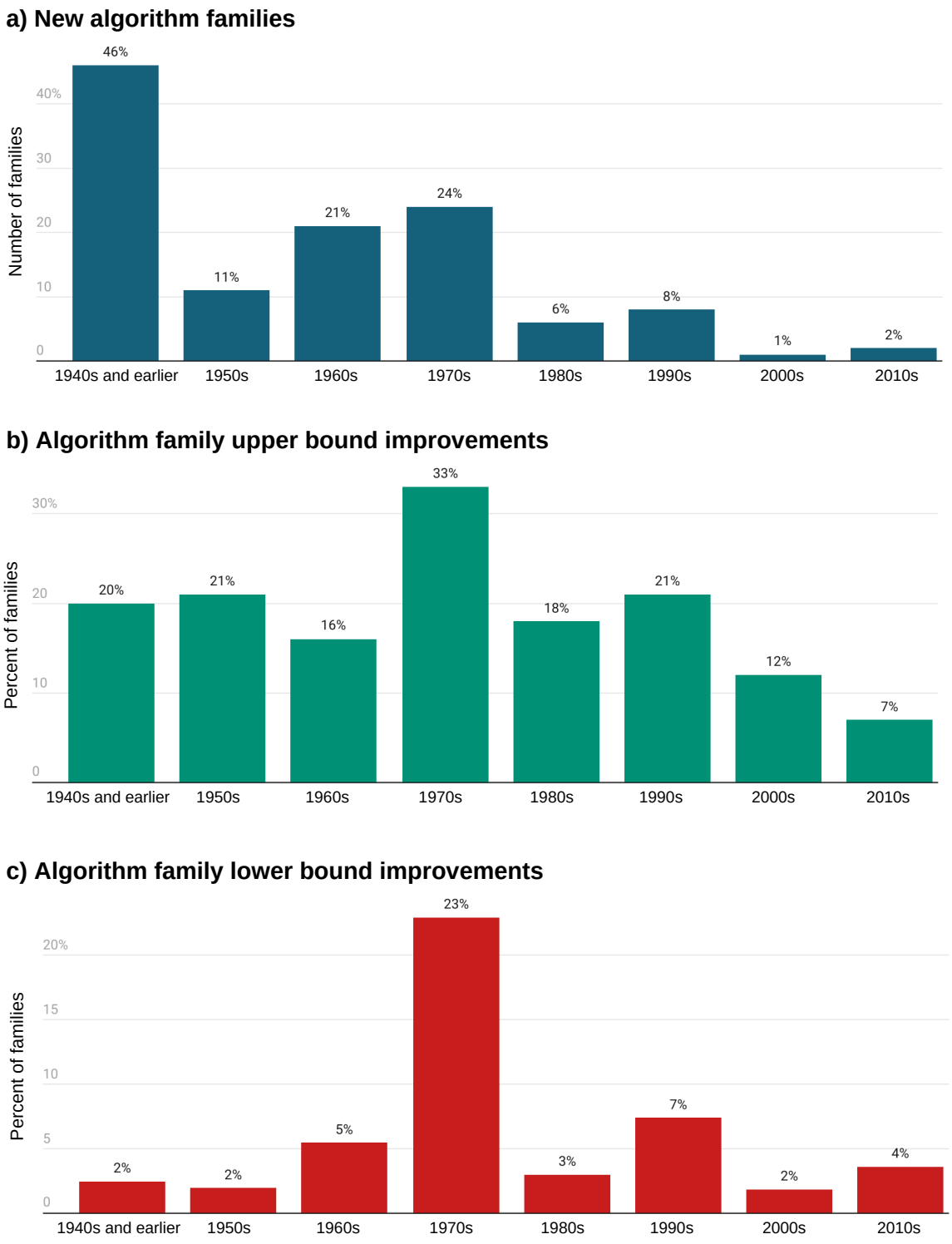


Figure 1-1: Historical trends of algorithm progress. **a** Number of new algorithm families discovered each decade. Share of known algorithm families whose **b** upper bound or **c** lower bound improved each decade. Figures **a** and **b** reproduced with permission from [48].

In order to prove a better lower bound, consider how much information is needed to distinguish the correct order of the elements. We can construct a decision tree, where each level asks a question that distinguishes two possibilities. There are $n!$ possibilities, which means there are $n!$ leaves of the tree. Then the minimum height of the tree is $\log(n!)$, which (by Stirling's approximation) is at least $\Omega(n \log n)$.

$\Omega(n \log n)$ is an improvement on $\Omega(n)$, and it is actually the best, or optimal, lower bound. One example of a sorting algorithm taking $O(n \log n)$ time is mergesort. In mergesort, the set of elements to be sorted is repeatedly divided, sorted in smaller chunks, and recombined (merged) together. Since mergesort takes $O(n \log n)$ time, and the sorting lower bound is $\Omega(n \log n)$, mergesort is an asymptotically optimal algorithm for sorting.

1.2 History

By the 1960s, many algorithm families had already been created and work on improving them (in the traditional sense, through upper-bound work) was well underway [48]. Figure 1-1 shows the trends of algorithm progress over time. As shown in Figure 1-1b, improvements in algorithm families peaked in the 1970s, with 33% of the families that existed in that decade making gains. There has been a steady decrease since then, and 7% of algorithm families improved in the 2010s.

Work on lower bounds, presented in Figure 1-1c, did not truly begin until around the 1970s, during which there was a huge spike in lower bound improvements: 23% of families saw lower bound improvements in the 1970s, compared to just 5% in the previous decade. In the decades since, the pattern of lower bound improvements is roughly comparable to the pattern of new algorithm families created, suggesting that lower-bound work continues steadily.

Chapter 2

Results

Our work follows closely that of Sherry and Thompson, and we use the same idea of *families* of algorithms that all solve the same computing problem. Here we also use the terms *algorithm family* and *problem* nearly interchangeably.

In keeping with the conventions typically followed by algorithm designers, we shall use Ω and O notation for lower- and upper-bounds respectively, to express running time as a function of input size. This type of notation ignores constant factors and lower-order terms inherent in implementation, which allows us to focus on how an algorithm scales up to large problem sizes and to more easily compare different algorithms. For example, a lower bound of $\Omega(n)$ means that every algorithm must take time at least proportional to n, and an upper bound of $O(n)$ means that there exists an algorithm that takes time at most proportional to n.

As an example, mergesort, described earlier, has an upper bound on running time of $O(n \log n)$. This means that, at a large enough problem size, there is some constant c for which the function $cn \log n$ is always greater than the running time of mergesort. Practically speaking, we can treat all algorithms within the same time complexity as scaling similarly to each other, up to constant factors.

2.1 Examples

In order to get a better picture of what progress looks like, we can visualize how upper and lower bounds for individual algorithm families converge over time. Figure 2-1 plots the evolution of a few different algorithm families.

2.1.1 Matrix Chain Multiplication

The first, Matrix Chain Multiplication, is the problem of finding the most efficient way to multiply a given sequence of matrices. More specifically, we want to determine the order of multiplications that requires the fewest arithmetic operations (performing the actual matrix multiplications is a separate problem). This is useful in compilers for code optimization, and in databases for query optimization.

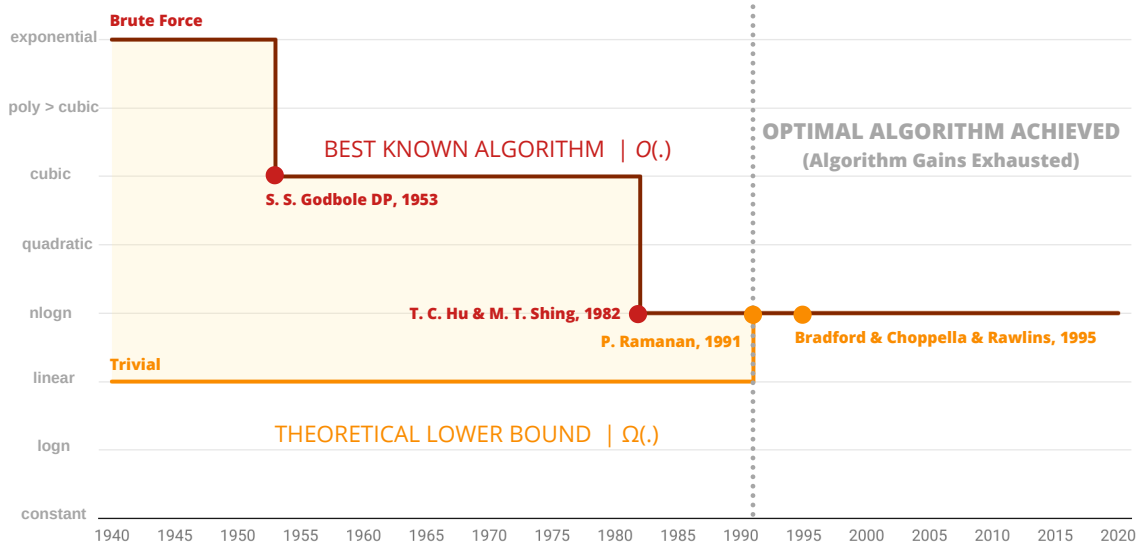
Suppose we have matrices $\mathbf{A}$ which is 6 by 5 (6 rows by 5 columns), $\mathbf{B}$ which is 5 by 8, and $\mathbf{C}$ which is 8 by 4. We want to find the fastest way to compute the product $\mathbf{ABC}$. Since matrix multiplication is associative, the two options are to compute $\mathbf{AB}$ then multiply by $\mathbf{C}$, or $\mathbf{BC}$ first and multiply $\mathbf{A}$ by that intermediate product. In other words, we could compute $(\mathbf{AB})\mathbf{C}$ or $\mathbf{A}(\mathbf{BC})$.

These two calculations require different numbers of computations. Computing $(\mathbf{AB})\mathbf{C}$ requires $6 \times 5 \times 8 + 6 \times 8 \times 4 = 432$ computations, while computing $\mathbf{A}(\mathbf{BC})$ requires $5 \times 8 \times 4 + 6 \times 5 \times 4 = 280$ computations. The second option requires fewer arithmetic operations and therefore is the correct answer.

As illustrated in 2-1a, there were several improvements over the years. The brute force method, which is simply to check all possibilities, takes exponential time. In 1953, Richard Bellman [12] invented dynamic programming, a powerful technique of dividing a large problem into smaller intermediate problems to optimize; in 1973 Godbole [22] showed how to use this technique to get an $O(n^3)$ algorithm. In 1981, Hu and Shing [27] used a more complicated technique involving triangulations of regular polygons to obtain an $O(n \log n)$-time algorithm.

On the lower bound side, Ramanan [39] in 1991 proved an $\Omega(n \log n)$ lower bound for the Matrix Chain Multiplication problem by first proving a lower bound for poly-

a) Matrix chain multiplication progress



b) Sequence alignment progress

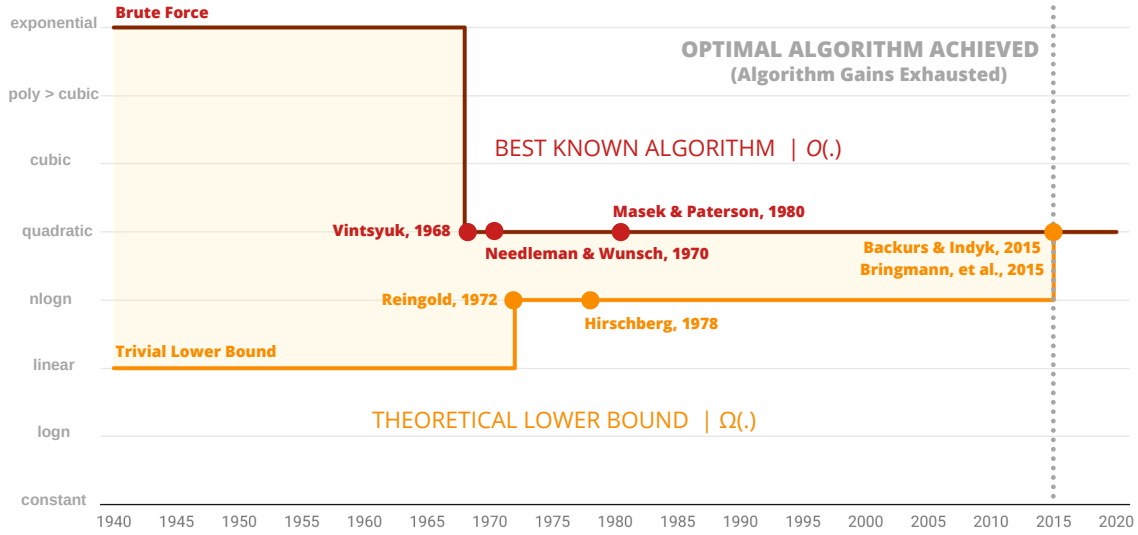


Figure 2-1: The upper and lower bounds of algorithm families over time. **a** Matrix chain multiplication. **b** Sequence alignment. A subsequent marker at the same complexity but at a later date indicates either that an independent discovery of the same bound was made, or that an improvement was made but it was not large enough to jump between the labeled categories.

gon triangulation and then relating that lower bound to Matrix Chain Multiplication. Shortly after that, in 1995, Bradford et al. [14] gave a different, more direct proof of the same lower bound, by proving that a class of solutions to Matrix Chain Multiplication requires a large input size. Both of these bounds match the best upper bound, which means that the Matrix Chain Multiplication problem is now at an optimal time complexity of $O(n \log n)$. This is shown in Figure 2-1a as the convergence of the upper and lower bound lines. Because this algorithm family's upper and lower bounds have converged, there are no longer any asymptotic improvements possible for Matrix Chain Multiplication.

2.1.2 Sequence Alignment

We can follow a similar storyline for Sequence Alignment (also known as Edit Distance). The problem here essentially is to find the number of single-character edits required to transform one string (or sequence) into another. For example, transforming `HUMPTY` to `DUMPTY` simply requires replacing the `H` with `D`, or one edit. Transforming `PREDICTION` to `PREDILECTION` requires two insertions. The name Sequence Alignment is due to its application in bioinformatics, in which researchers assess different DNA sequences to search for similarities between subsequences.

While an $O(n^2)$-time algorithm for Edit Distance due to Vintsyuk [53] was known as early as 1968, it was not until 2015 that this was shown to be the optimal algorithm when two independent teams proved the matching lower bound. Also noteworthy, though not explicitly noted in this diagram, is that the $\Omega(n^2)$ lower bound here is dependent on a conjecture, the Strong Exponential Time Hypothesis (SETH). This conjecture is largely believed to be true by the computer science community [52] and thus we treat it that way. [1]

[1] See the Discussion and Methods sections for more details on how we treat these types of conditional lower bounds.

2.2 Summary of Results

2.2.1 Overall Optimality

In total, we analyzed 113 distinct algorithm families. In keeping with [48], we only studied worst-case complexities of exact algorithm families.[2] We repeated the data collection process behind Figure 2-1 for each of these algorithm families: we recorded lower bounds for each of the algorithm families from [48], and compared them to the corresponding upper bounds.

In order to measure the progress gap between upper and lower bounds, we measured the gap between the best upper and lower bounds for each family. Specifically, we computed the quotient between the best asymptotic upper and lower bounds and then classified the algorithm families into five categories:

- **Optimal**. Problems in this category have a constant difference between their upper and lower bounds, and we say that they have matching upper and lower bound. Asymptotically optimal algorithms exist for these problems.

- **Sublinear difference**: $(1,\, n)$. This category includes anything asymptotically sublinear, which includes problems with a polylogarithmic difference. The term "polylogarithmic" refers to something which is polynomial in $\log n$. For instance, $\log^3(n)$ is polylogarithmic.

- **Linear difference**: $[n,\, n \log^k n]$. The "linear" category accepts anything within a polylogarithmic factor above linear, such as $O(n \log n)$ or $O(n \log^2 n)$.

- **Polynomial difference**: $(n \log^k n,\ n^k]$. Here we only consider polynomials with an exponent greater than 1, such as $O(n^{1.5})$ or $O(n^3)$.

- **Exponential difference**: $(n^k,\, \infty)$. This category includes problems where the upper bound is superpolynomial or greater (which includes exponential), and the lower bound is polynomial or less.

[2] See Methods for more on the criteria to be an included algorithm family.

To demonstrate the contrast between these categories, consider two fictional problems, A and B. If problem A has an upper bound of $O(n^4)$ and a lower bound of $\Omega(n^2)$, then it would have a gap of $O(n^2)$, which would be classified as polynomial. On the other hand, problem B with an upper bound of $O(n \log n)$ and a lower bound of $\Omega(n)$ would have a gap of $O(\log n)$, which would be classified as having a sublinear difference. For some perspective on how these values scale, consider a problem size of $n = 100$. The quantity n^2 at $n = 100$ is 10000, while the quantity $\log n$ at the same value of n is 6.6 (assuming base-2)—a 1500× multiplicative difference.

Figure 2-2a presents the resulting data from grouping the problems this way. 64% of the analyzed problems are already at optimal, and a further 15% have only a sublinear or polylogarithmic multiplicative potential improvement.

In other words, more than three-quarters of the algorithm families in our dataset are very nearly at optimal, with no further improvement possible of a polynomial or greater size.

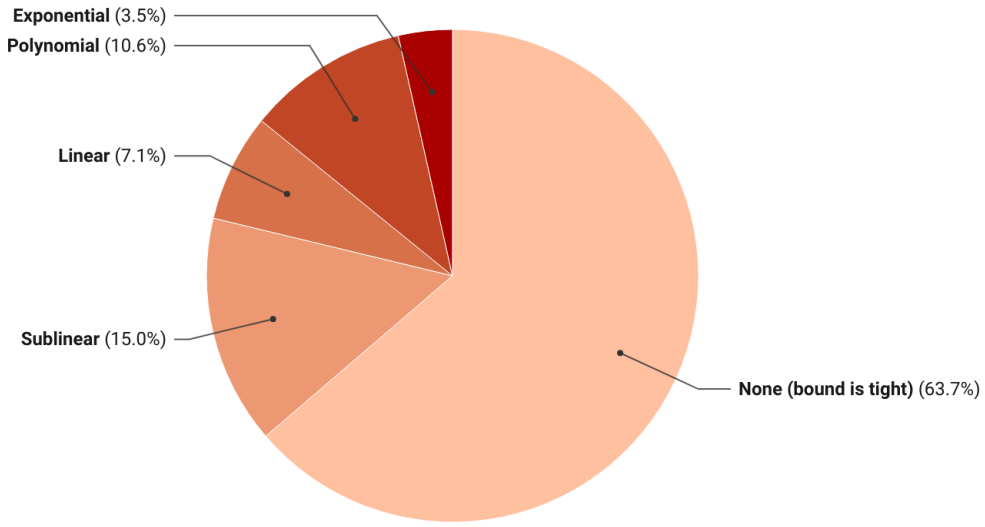
2.2.2 Remaining Opportunities for Algorithmic Improvement

The rest of Figure 2-2 quantifies the remaining opportunities for improvement. Figure 2-2b illustrates where there is still possibility for exponential improvement, while 2-2c provides a more "zoomed-in" view of problems with polynomial running times.

The leftmost column of squares in 2-2b contains all the algorithm families with an exponential upper bound. The topmost square in that column displays the number of families with a matching exponential lower bound, and the squares below list the number of families with differently categorized lower bounds. These categories were chosen so that different categories represent exponentially different time complexities. In total, out of the 113 problems analyzed, only 4 have upper and lower bounds in different categories. The other vast majority, 109 algorithm families, cannot jump between upper or lower bound categories and therefore have *no further exponential improvements possible.*

However, the majority of these are contained in the rather large category of having both a polynomial upper bound and a polynomial lower bound. For example,

a) Gaps between upper and lower bounds



b) Bound categorizations

Lower bounds \ Upper bounds	Exponential	Polynomial or Linear	Sublinear
Exponential	22		
Polynomial or Linear	4	82	
Sublinear	0	0	5

c) Polynomial bound categorizations

Lower bounds \ Upper bounds	$> n^3$	n^3	n^2	$n \log n$	n
$> n^3$	0				
n^3	0	0			
n^2	0	1	6		
$n \log n$	0	0	0	12	
n	2	10	11	8	32

Figure 2-2: **a** Gaps between upper and lower bounds, grouped into five categories. For the purposes of this graph, "linear" includes polynomials within a polylogarithmic factor above $O(n)$, while "polynomial" is restricted to polynomials with degree strictly greater than 1; "exponential" includes anything superpolynomial or greater; and we count NP-hard and NP-complete problems as having a "tight" bound. **b** Breakdown of upper and lower bounds into three categories for each: exponential, polynomial or linear, and sublinear. **c** Breakdown of upper and lower bounds within the polynomial category into five further divisions for each.

$O(n \log n)$ and $O(n^4)$ are both polynomial but have substantially different growth rates. Therefore, we present a more granular division of these 82 problems in Figure 2-2c. We can again see that most problems here have matching upper and lower bounds. 50 out of these 82 problems are optimal: 6 problems with a complexity of $O(n^2)$, 12 at $O(n \log n)$, and 32 at $O(n)$.

Chapter 3

Discussion

3.1 The Decline of Algorithm Progress

Our results suggest that many algorithms are already optimal, and no more progress is possible on the exact versions of these problems. This might explain why upper-bound progress in exact algorithms has seemed to be slowing in recent decades.

Figure 1-1b, reproduced here with permission from [48], displays the percentage of algorithms whose upper bounds improve each decade, in which there is a clear drop-off in the decades following the 1970s. Sherry and Thompson raise as one potential explanation that some algorithms are already optimal, leaving little room for further improvement. Our data lend support to this hypothesis.

Of course, there are many factors that were completely outside of the scope of this survey, which only considered exact algorithms in textbooks. For one, algorithms that have made it into published textbooks are likely to be very well-studied and therefore have more complete work done already. Also, this is a relatively small scope compared to the gigantic field of computer science, and there is certainly progress possible in fields like approximate, parallel, and quantum algorithms (just to name a few), which are explicitly not included in this dataset.

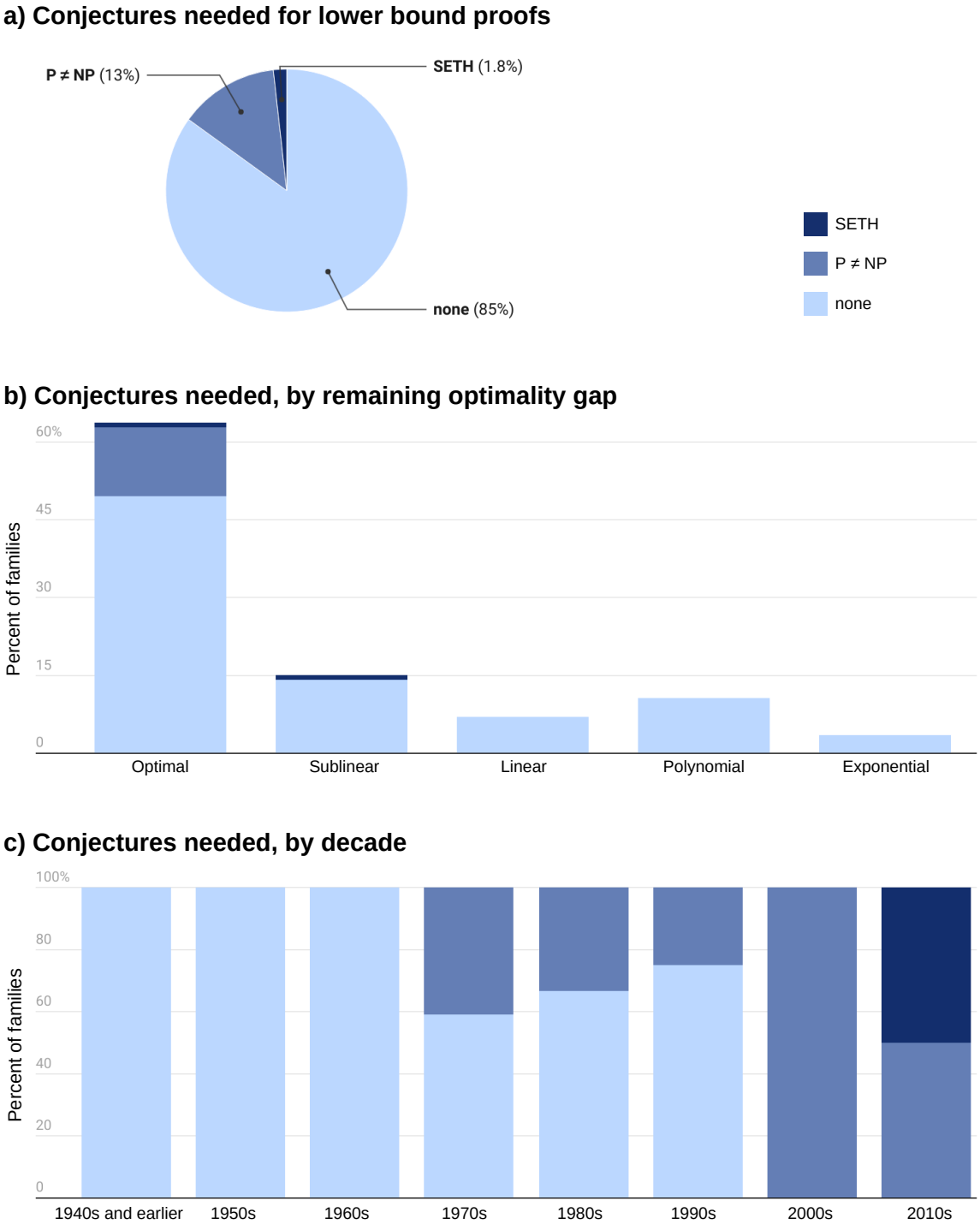


Figure 3-1: Use of conjectures in proving lower bounds. **a** Share of lower bounds requiring conjectures for their proof. **b** Gaps between upper and lower bounds, grouped as in Figure 2-2, showing the share of lower bounds that require conjectures. **c** Percent of algorithm families with lower bound improvements requiring each conjecture, by decade.

3.2 The Evolution of Lower-Bound Techniques

Many algorithm families have been optimal for the past few decades, but researchers just did not have the techniques to prove the matching lower bounds. One such technique is the use of conjectures, or unproven assumptions that are nevertheless widely believed to be true [52]. Some of the lower bound proofs found in this survey paper require conjectures, either P $\neq$ NP or SETH, the Strong Exponential Time Hypothesis. We call such lower bounds conditional lower bounds, since they are conditional on a conjecture.

Our data suggest that 15% of lower bounds are conditional. These conjectures can be very powerful; as shown in Figure 3-1b, in the algorithm families analyzed, the presence of a conjecture means the algorithm family is either optimal or near-optimal. Figure 3-1c also shows how work on conditional lower bounds has been increasing over the decades, compared to regular unconditional lower bounds. Lower bounds involving SETH did not even appear until the 2010s, and *all* of the lower bounds proven after the 2000s were conditional. This suggests that these conjectures are becoming a more powerful tool for proving lower bounds.

Chapter 4

Methods

4.1 Overview

We began with going through a set of exact algorithm families, formulated by [48] through careful examination of important algorithms textbooks. In our data gathering phase, we compiled a list of lower bounds for these algorithm families. Specifically, for each algorithm family, we searched for proven lower bounds in the literature and then recorded the lower bound itself, the notation for the lower bound, the paper proving it, and the year of publication.

We also recorded some additional information about the lower bound. Some lower bounds are inherent in the problem statement itself—for example, the amount of time it takes to fully read in an input—and these bounds we refer to as "trivial." We also noted whether it is applicable only to a restricted version of the problem, such as a special case or a restricted computation model; such lower bounds are interesting to consider but may not be applicable in overall comparisons to upper bounds. Finally, we also needed to consider whether the bound is conditional on some unproven conjecture, and if so we recorded that conjecture. The two conjectures that we ended up including are P $\neq$ NP and the Strong Exponential Time Hypothesis (SETH).

To give some background, the terms P and NP are computational complexity classes, meaning that they refer to sets of problems with certain computational char-

acteristics. The class P contains every problem that can be solved in polynomial time, while the class NP contains every problem whose solution can be verified, or checked, in polynomial time. The P versus NP problem asks whether every problem that can be solved "quickly"—in polynomial time—can also be checked "quickly." A problem being NP-hard means that it is at least as hard as every problem in NP. In terms of lower bounds, a problem being NP-hard means that there is an exponential lower bound *if and only if* P $\neq$ NP. The Strong Exponential Time Hypothesis (SETH) is a stronger statement than P $\neq$ NP. Either of these conjectures, if true, would have strong implications for the existing structure of complexity classes.

In general, we started by reading through the papers in [48] that prove upper bounds to get some background and understanding on the problem. We then performed searches in Google Scholar for relevant papers in the field for lower bounds to the problem. This began with a search (and read-through) among the papers that cite a seminal upper bounds paper, and then looking through papers that cite, or are cited by, any promising looking papers. We repeated this process until we got a good sense of the lower bounds that exist and felt confident that we had found the papers that prove the relevant lower bounds (and lower bound improvements), if any exist. It is notable that in some cases the only lower bound reported is the trivial lower bound, either because the upper bound matches or almost matches the lower bound, or because there simply is not very much work done on the lower bound.

4.2 Algorithm Family Refinements

There are some problems that are so hard that the primary approaches to these problems are approximations. While approximation algorithms is a rich subfield in algorithms, they are outside of the scope of this paper, which focuses on the progress in exact algorithms. We do not yet have a framework to compare bounds for exact and approximate algorithms. However, we did not want to exclude problems that were clearly important, by the judgement of textbook authors and paper authors and by our own experiences. After some discussion we decided to include problems that

have at least an exact brute-force solution. This excluded problems such as Integral Equations, which did not have any general-purpose exact algorithm.

In a few cases, multiple algorithms solving distinct versions of a problem were grouped together as the same algorithm family. We made sure to clarify potentially ambiguous problem statements and to separate out problems that belonged in separate families.

For consistency, we parameterize by problem (input) size in all cases. This does go against convention for some problems, namely matrix-related algorithms, where conventionally n is the side length of the matrix; by our terminology this is actually the square root of the problem size. This also occurs for some graph problems, or other problems where a matrix representation is used.

4.3 Analysis

In the process of gathering our data, there were some bounds that did not fit neatly into our broad categorizations. These ended up being algorithms that were quasipolynomial, which is $\exp(\log^c n)$ for some constant $c > 1$. They fall in an odd gap between categories, slower than any polynomial-time algorithm, but not as slow as any exponential-time algorithm. We ended up categorizing these all as "exponential" for the purpose of our analysis.

Appendix A

Lower Bound Data

Below is a list of all the algorithm families included in this study and the lower bounds found for them. See [48] for more details on the families.

As noted in the Methods, n represents the problem size. Problems marked with an asterisk (*) in the Notes are ones where the problem parameter is conventionally something different. For example, for matrix problems, n is more conventionally the side length of the matrix, while the problem size is actually n^2. Since we report in terms of problem size, we say that the lower bound of Matrix Multiplication is linear in the problem size, not quadratic. Citations in the Notes are additional sources that are interesting and relevant but are not necessarily lower-bound papers.

Algorithm Family	**Lower Bound(s)**	**Year**	**Source**	**Notes**
Sorting	$\Omega(n \log n)$	1721	[1]	
Integer Sorting	$\Omega(n)$	–	trivial	
Matrix Chain Multiplication	$\Omega(n \log n)$	1994	[39]	
	$\Omega(n \log n)$	1995	[14]	
Longest Common Subsequence	$\Omega(n)$	–	trivial	
Maximum Flow	$\Omega(n^2)$	2015	[1]	SETH
Matrix Multiplication	$\Omega(n)$	–	trivial	[6], *
3-Graph Coloring	exponential	1973	[32]	NP-hard
4-Graph Coloring	exponential	1973	[32]	NP-hard

[1] Stirling's approximation

Algorithm Family	**Lower Bound(s)**	**Year**	**Source**	**Notes**
Linear System of Equations	$\Omega(n)$	–	trivial	*
Linear Programming	$\Omega(n)$	–	trivial	*
Line Segment Intersection	$\Omega(n \log n)$	1997	[10]	
Convex Hull	$\Omega(n \log n)$	1975	[46]	[55]
Strongly Connected Components	$\Omega(n)$	1972	[50]	
Minimum Spanning Tree	$\Omega(n)$	–	trivial	[23], [24]
Closest Pair Problem	$\Omega(n \log n)$	1975	[46]	
All-pairs Shortest Path	$\Omega(n^2 \log n)$	1977	[54]	
First Category Integer Factoring	$\Omega(n)$	–	trivial	
Second Category Integer Factoring	$\Omega(n)$	–	trivial	[9]
LU Decomposition	$\Omega(n)$	–	trivial	*
Informed Search	$b^{d/2}$	–	trivial	
String Search	$3n(1 - o(1))$	1994	[17]	
Sequence Alignment	$\Omega(n \log n)$	1972	[41]	
	$\Omega(n \log n)$	1978	[26]	
	$\Omega(n^2)$	2015	[15]	SETH
	$\Omega(n^2)$	2015	[7]	SETH
Joins	$\Omega(n + m)$	–	trivial	
Line Clipping	$\Omega(1)$	–	trivial	
NFA to DFA Conversion	$\Omega(2^{n/2})$	1996	[43]	
Multiplication	$\Omega(n)$	–	trivial	
	$\Omega(n \log n)$	1969	[18]	online
Maximum Cardinality Matching	$\Omega(n)$	–	trivial	
Key Exchange	$\Omega(n)$	–	trivial	[13], [8], [49]
Mutual Exclusion	$\Omega(1)$	–	trivial	[34], [40]
SDD Systems Solvers	$\Omega(n)$	–	trivial	
Cycle Detection	$\Omega(n)$	1982	[45]	[16]
Generating Random Permutations	$\Omega(n)$	–	trivial	

Algorithm Family	**Lower Bound(s)**	**Year**	**Source**	**Notes**
Gröbner Bases	$\Omega(n)$	–	trivial	
Minimum value[2]	$\Omega(n)$	–	[3]	
All Permutations	$\Omega(n)$	–	trivial	
Huffman Encoding	$\Omega(n)$	–	trivial	
Nash Equilibria	$\Omega(n)$	–	trivial	
Maximum-weight Matching	$\Omega(n)$	–	trivial	
Constructing Eulerian Trails in a Graph	$\Omega(n)$	–	trivial	
Line Drawing	$\Omega(n)$	–	trivial	
Polygon Clipping	$\Omega(n \log n)$	1976	[47]	
Nearest Neighbor Search	$\Omega(n \log n)$	1975	[46]	
Coset Enumeration	long	1955	[38]	
Register Allocation	$\Omega(n)$	–	trivial	
Voronoi Diagrams	$\Omega(n \log n)$	1975	[46]	
Topological Sorting	$\Omega(n)$	–	trivial	
DFA Minimization	$\Omega(n)$	–	trivial	
Lowest Common Ancestor	$\Omega(n)$	–	trivial	[25]
Graph Edit Distance Computation	exponential	1994	[31]	NP-hard, [56]
Enumerating Maximal Cliques	exponential	1980	[30]	NP-hard
The Traveling-Salesman Problem	exponential	1972	[28]	NP-hard
2-D Elliptic Partial	$\Omega(n^2)$	1972	[44]	
3-D Elliptic Partial	$\Omega(n^2)$	1972	[44]	
Delaunay Triangulation	$\Omega(n \log n)$	1975	[46]	
De Novo Gene Assembly	$\Omega(n)$	–	trivial	
Subset-Sum	exponential	1972	[28]	NP-hard
Dependency Inference	exponential	1992	[35]	
BCNF Decomposition	exponential	1979	[11]	NP-hard, [51]
4NF Decomposition	exponential	1979	[11]	NP-hard

[2]Minimum value in each row of an implicitly-defined totally monotone matrix

Algorithm Family	**Lower Bound(s)**	**Year**	**Source**	**Notes**
Discovering Multivalued Dependencies	exponential	1992	[35]	
Disk Scheduling	$\Omega(n)$	–	trivial	
Vertex-Cover	exponential	1972	[28]	NP-complete
Parsing	$\Omega(n)$	–	trivial	[19]
Finding Frequent Item Sets	exponential	1992	[29]	P#-complete
Lossy Compression	$\Omega(n)$	–	trivial	
Factorization of Polynomials over Finite Fields	$\Omega(n)$	–	trivial	
Cryptanalysis of Linear Feedback Shift Registers	$\Omega(n)$	–	trivial	
Stable Marriage	$\Omega(n)$	–	trivial	*
Longest Path on Interval Graphs	$\Omega(n)$	–	trivial	
Maximum Subarray	$\Omega(n)$	–	trivial	
Constructing Suffix Trees	$\Omega(n)$	–	trivial	
Entity Resolution	$\Omega(n)$	–	trivial	
Longest Palindromic Substring	$\Omega(n)$	–	trivial	
Translating Abstract Syntax Trees into Code	$\Omega(n)$	–	trivial	
Graph Isomorphism Problem	$\Omega(n)$	–	trivial	
Digraph Realization	$\Omega(n)$	–	trivial	
Duplicate Elimination	$\Omega(n)$	–	trivial	
Matrix Factorization for Collaborative Filtering	$\Omega(n)$	–	trivial	*
MDPs for Optimal Policies	$\Omega(n)$	–	trivial	
Set-Covering	exponential	1972	[28]	NP-complete
Motif Search	exponential	2000	[5]	NP-hard
Link Analysis (Indegree)	$\Omega(n)$	–	trivial	
Link Analysis (Pagerank)	$\Omega(n)$	–	trivial	
Cyclopeptide Sequencing	$\Omega(n)$	–	trivial	

Algorithm Family	**Lower Bound(s)**	**Year**	**Source**	**Notes**
Point in Polygon	$\Omega(n)$	–	trivial	
Maximum Cut	exponential	1972	[28]	NP-complete
Minimum Wiener Connector	exponential	2015	[42]	NP-hard
Determinant using Integer Arithmetic	$\Omega(n)$	–	trivial	*
Integer Relation	$\Omega(n)$	–	trivial	
Sequence to Graph Alignment (Linear Gap Penalty)	$\Omega(n)$	–	trivial	
Logarithm Calculations	$\Omega(n)$	–	trivial	
Rod Cutting	$\Omega(n)$	–	trivial	
Transitive Reduction	$\Omega(n^2)$	1972	[4]	
Change-Making	$\Omega(2^n)$	1975	[33]	NP-complete
Turnpike Problem	$\Omega(n)$	–	trivial	
n-Queens Problem	$\Omega(2^n)$	2017	[21]	NP-hard
Median String	exponential	2005	[37]	NP-complete
Frequent Words	$\Omega(n)$	–	trivial	
Secret Sharing	$\Omega(tn)$	–	trivial	
Polynomial Interpolation	$\Omega(n)$	–	trivial	*
Greatest Common Divisor	$\Omega(n)$	–	trivial	[36]
Weighted Activity Selection	$\Omega(n \log n)$	1976	[47]	
	$\Omega(n \log n)$	1978	[20]	
Single-interval Scheduling Maximization (Unweighted)	$\Omega(n)$	–	trivial	
Self-Balancing Trees Creation	$\Omega(n \log n)$	1972	[2]	
Self-Balancing Trees Insertion	$\Omega(\log n)$	1972	[2]	
Self-Balancing Trees Deletion	$\Omega(\log n)$	1972	[2]	
Self-Balancing Trees Search	$\Omega(\log n)$	1972	[2]	
Deadlock Avoidance	$\Omega(n)$	–	trivial	
Page Replacements	$\Omega(n)$	–	trivial	
Recovery	$\Omega(n)$	–	trivial	

Bibliography

[1] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018.

[2] George M Adel'son-Vel'skii and Evgenii Mikhailovich Landis. An algorithm for organization of information. In *Doklady Akademii Nauk*, volume 146, pages 263–266. Russian Academy of Sciences, 1962.

[3] Alok Aggarwal, Maria M Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

[4] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[5] Tatsuya Akutsu, Hiroki Arimura, and Shinichi Shimozono. On approximation algorithms for local multiple alignment. In *Proceedings of the fourth annual international conference on Computational molecular biology*, pages 1–7, 2000.

[6] Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 580–591. IEEE, 2018.

[7] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58, 2015.

[8] Boaz Barak and Mohammad Mahmoody-Ghidary. Merkle puzzles are optimal—an o (n 2)-query attack on any key exchange from a random oracle. In *Annual International Cryptology Conference*, pages 374–390. Springer, 2009.

[9] Connelly Barnes. Integer factorization algorithms. *Department of Physics Oregon State University*, 2004.

[10] Julien Basch, Leonidas J Guibas, and GD Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 469–471, 1997.

[11] Catriel Beeri and Philip A Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems (TODS)*, 4(1):30–59, 1979.

[12] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[13] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. Perfectly-secure key distribution for dynamic conferences. In *Annual international cryptology conference*, pages 471–486. Springer, 1992.

[14] Phillip G Bradford, Venkatesh Choppella, and Gregory JE Rawlins. Lower bounds for the matrix chain ordering problem. In *Latin American Symposium on Theoretical Informatics*, pages 112–130. Springer, 1995.

[15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97. IEEE, 2015.

[16] Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Cycle detection, order finding and discrete log with jumps. In *ICS*, pages 284–297. Citeseer, 2011.

[17] Richard Cole. Tight bounds on the complexity of the boyer–moore string matching algorithm. *SIAM Journal on Computing*, 23(5):1075–1091, 1994.

[18] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[19] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[20] Michael L Fredman and Bruce Weide. On the complexity of computing the measure of ∪[ai, bi]. *Communications of the ACM*, 21(7):540–544, 1978.

[21] Ian P Gent, Christopher Jefferson, and Peter Nightingale. Complexity of n-queens completion. *Journal of Artificial Intelligence Research*, 59:815–848, 2017.

[22] Sadashiva S Godbole. On efficient computation of matrix chain products. *IEEE Transactions on Computers*, 100(9):864–866, 1973.

[23] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[24] Mohamed Haouari and Jouhaina Siala Chaouachi. Upper and lower bounding strategies for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 171(2):632–647, 2006.

[25] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

[26] Daniel S. Hirschberg. An information theoretic lower bound for the longest common subsequence problem. *Rice University ECE Technical Report*, (7705), 1977.

[27] TC Hu and MT Shing. Computation of matrix chain products. part ii. *SIAM Journal on Computing*, 13(2):228–251, 1984.

[28] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[29] Toshinobu Kashiwabara, Sumio Masuda, Kazuo Nakajima, and Toshio Fujisawa. Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc graph. *Journal of algorithms*, 13(1):161–174, 1992.

[30] Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[31] Chih-Long Lin. Hardness of approximating graph transformation problem. In *International Symposium on Algorithms and Computation*, pages 74–82. Springer, 1994.

[32] László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12. Utilitas Mathematica Publishing, 1973.

[33] George S Lueker. *Two NP-complete problems in nonnegative integer programming*. Princeton University. Department of Electrical Engineering, 1975.

[34] Mamoru Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems (TOCS)*, 3(2):145–159, 1985.

[35] Heikki Mannila and Kari-Jouko Räihä. On the complexity of inferring functional dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[36] Yishay Mansour, Baruch Schieber, and Prasoon Tiwari. A lower bound for integer greatest common divisor computations. *Journal of the ACM (JACM)*, 38(2):453–471, 1991.

[37] François Nicolas and Eric Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of discrete algorithms*, 3(2-4):390–415, 2005.

[38] Petr Sergeevich Novikov. Algorithmic unsolvability of the word problem in group theory. *Journal of Symbolic Logic*, 23(1), 1958.

[39] Prakash Ramanan. A new lower bound technique and its application: Tight lower bound for a polygon triangulation problem. *SIAM Journal on Computing*, 23(4):834–851, 1994.

[40] Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems (TOCS)*, 7(1):61–77, 1989.

[41] Edward M Reingold. On the optimality of some set algorithms. *Journal of the ACM (JACM)*, 19(4):649–659, 1972.

[42] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. The minimum wiener connector problem. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1587–1602, 2015.

[43] Kai Salomaa and Sheng Yu. Nfa to dfa transformation for finite languages. In *International Workshop on Implementing Automata*, pages 149–158. Springer, 1996.

[44] Martin H Schultz. The computational complexity of elliptic partial differential equations. In *Complexity of Computer Computations*, pages 73–83. Springer, 1972.

[45] Robert Sedgewick, Thomas G Szymanski, and Andrew C Yao. The complexity of finding cycles in periodic functions. *SIAM Journal on Computing*, 11(2):376–390, 1982.

[46] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975.

[47] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 208–215. IEEE, 1976.

[48] Yash Sherry and Neil Thompson. How fast do algorithms improve. Technical report, Mimeo, 2020.

[49] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. A lower-bound of complexity for rsa-based password-authenticated key exchange. In *European Public Key Infrastructure Workshop*, pages 191–205. Springer, 2005.

[50] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[51] Don-Min Tsou and Patrick C Fischer. Decomposition of a relation scheme into boyce-codd normal form. *ACM SIGACT News*, 14(3):23–29, 1982.

[52] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[53] Taras K Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[54] Andrew C Yao, David M Avis, and Ronald L Rivest. An ω (n2 log n) lower bound to the shortest paths problem. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 11–17, 1977.

[55] Andrew Chi-Chih Yao. A lower bound to finding convex hulls. *Journal of the ACM (JACM)*, 28(4):780–787, 1981.

[56] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.