AN INTRODUCTION TO TIME COMPLEXITY AND NP-COMPLETE

Upender Yadav
Dronacharya college of engineering,
Gurgaon, Haryana, India
Upenderyadav1424@gmail.com
Ph: - +91 9810537193

Vikash Verma
Dronacharya college of engineering,
Gurgaon, Haryana, India
vermavikash153@gmail.com
ph: - +91 8802806339

Parmod Saha
Dronacharya college of engineering,
Gurgaon, Haryana, India
parmodsah000@gmail.com
ph: - +91 9910852711

ABSTRACT:

The time complexity of an algorithm is the amount of computer time required by an algorithm to run to completion. Computational time complexity analyzes of evolutionary algorithms (EAs) have been performed since the mid-nineties. The first results were related to very simple algorithms, such as the (1+1)-EA, on toy problems. The objective of this paper is to review the time complexity knowledge seen in research work devoted on synthesis, optimization, and effectiveness of various sorting algorithms. We will examine different sorting algorithms in the lines and try to discover the tradeoffs between them.

KEY WORDS:

Algorithm analysis, Sorting algorithm, Computational Complexity notation

INTRODUCTION:

It all started with a machine. In 1936, Turing developed his theoretical computational model. He based his model on how he perceived mathematicians think. As digital computers were developed in the 40's and 50's, the Turing machine proved itself as the right theoretical model for computation. Quickly though we discovered that the basic Turing machine model fails to account for the amount of time or memory needed by a computer, a critical issue today but even more so in those early days of computing. The key idea to measure time and space as a function of the length of the input came in the early 1960's by Hartmanis and Stearns. And thus computational complexity was born. In the early days of complexity, researchers just tried understanding these new measures and how they related to each other. We saw the first notion of efficient computation by using time polynomial in the input size. This led to complexity's most important concept, NP-completeness, and its most fundamental question, whether P = NP.

The work of Cook and Karp in the early 70's showed a large number of combinatorial and logical problems were NP-complete, i.e., as hard as any problem computable in nondeterministic polynomial time. The P = NP question is equivalent to an efficient solution of

any of these problems. In the thirty years hence this problem has become one of the outstanding open questions in computer science and indeed all of mathematics. In the 70's we saw the growth of complexity classes as researchers tried to encompass different models of computations. One of those models, probabilistic computation, started with a probabilistic test for primality, led to probabilistic complexity classes and a new kind of interactive proof system that itself led to hardness results for approximating certain NP-complete problems.

In the 80's we saw the rise of finite models like circuits that capture computation in an inherently different way. A new approach to problems like P = NP arose from these circuits and though they have had limited success in separating complexity classes, this approach brought combinatorial techniques into the area and led to a much better understanding of the limits of these devices from computations and most recently a deterministic algorithm for the original primality problem.

In the 90's we have seen the study of new models of computation like quantum computers and propositional proof systems. Tools from the past have greatly helped our understanding of these new areas.

(1) ALGORITHM ANALYSIS:

- An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.
- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)
- What do we mean by running time analysis?
  - Determine how running time increases as the size of the problem increases

(1.1) Types of Algorithm Analysis:

- Worst case
  a) Provides an upper bound on running time
  b) An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- Best case
  a) Provides a lower bound on running time
  b) Input is the one for which the algorithm runs the fastest

LOWER BOUND <= RUNNING TIME <= UPPER BOUND

- Average case
    a) Provides a prediction about the running time
    b) Assumes that the input is random

(2) NP-COMPLETENESS:

It was in the early 1970's that complexity theory first flowered, and came to play a central role in computer science. It did so by focusing on one fundamental concept and on the results and ideas stemming from it. This concept was NP-completeness and it has proved to be one of the most insightful and fundamental theories in the mathematics of the last half century. NP-completeness
capturesthecombinatorialdifficultyofanumberofcentralproblemswhichresistedefficientsolution and provides a method for proving that a combinatorial problem is as intractable as any NP problem. By the late 1960's, a sizable class of very applicable and significant problems which resisted polynomial time solution was widely recognized. These problems are largely optimization problems such as the traveling salesman problem, certain scheduling problems, or linear programming problems. They all have a very large number of possible solution where there is no obvious way to find an optimal solution other than a brute force search. As time passed and much effort was expended on attempts at efficiently solving these problems, it began to be suspected that there was no such solution. However, there was no hard evidence that this was the case nor was there any reason to suspect that these problems were in any sense difficult for the same reasons or in the same ways. The theory of NP-completeness provided precisely this evidence. Proving a problem in NP to be NP-complete tells us that it is as hard to solve as any other NP problem. Said another way, if there is any NP-complete problem that admits an efficient solution then every NP problem does so. The question of whether every NP problem has an efficient solution has resisted the efforts of computer scientists since 1970. It is known as the P versus NP problem and is among the most central open problems of mathematics. The fact that a very large number of fundamental problems have been shown to be NP-complete and that the problem of proving that P is not NP has proved to be so difficult has made this problem and the connected theory one of the most celebrated in contemporary mathematics. The P = NP problem is one of the seven Millennium Prize Problems and solving it brings a $1,000,000 prize from the Clay Mathematics Institute. Quite surprisingly, one of the earliest discussions of a particular NP-complete problem and the implications of finding an efficient solution came from Kurt G¨odel. In a 1956 letter to von Neumann [Har86, Sip83] G¨odel asks von Neumann about the complexity of what is now known to be an NP-complete problem concerning proofs in first-order logic and asks if the problem can be solved in linear or quadratic time. In fact, Go¨del seemed quite optimistic about finding an efficient solution. He fully realized that doing so would have significant consequences. It is worth noting that in about the same period there was considerable effort by Russian mathematicians working on similar combinatorial problems trying to prove that brute force was needed to solve them. Several of these problems eventually turned out to be NP-complete as well [Tra64]. The existence of NP-

complete problems was proved independently by Stephen Cook in the United States and Leonid Levin in the Soviet Union. Cook, then a graduate student at Harvard, proved that the satisfiability problem is NP-complete [Coo71]. Levin, a student of Kolmogorov at Moscow State University, proved that a variant of the tiling problem is NP-complete [Lev73]. Researchers strove to show other interesting, natural problems NP-complete. Richard Karp, in a tremendously influential paper [Kar72], proved that eight central combinatorial problems are all NP-complete. These problems included the the clique problem, the independent set problem, the set cover problem, and the traveling salesman problem, among others. Karp's paper presented several key methods to prove NP-completeness using reductions from problems previously shown to be NP-complete. It set up a general framework for proving NP completeness results and established several useful techniques for such proofs. In the following years, and continuing until today, literally thousands of problems have been shown to be NP complete. A proof of NP-completeness has come to signify the (worst case) intractability of a problem. Once proved NP-complete, researchers turn to other ways of trying to solve the problem, usually using approximation algorithms to give an approximate solution or probabilistic methods to solve the problem in "most" cases. Another fundamental step was taken around 1970 by Meyer and Stockmeyer [MS72], [Sto76]. They defined the polynomial hierarchy in analogy with the arithmetic hierarchy of Kleene. This hierarchy is defined by iterating the notion of polynomial jump, in analogy with the Turing jump operator. This hierarchy has proven useful in classifying many hard combinatorial problems which do not lie in NP. It is explored in more detail in Section 4.2. Of course, all problems in the polynomial hierarchy are recursive and in fact very simple problems within the vast expanse of all recursive sets. So are there natural problems which are recursive and are not captured by the hierarchy? The answers is yes and results in the exploration of several important larger complexity classes which contain the polynomial hierarchy. One such class is PSPACE, those problems which can be solved using work space which is of polynomial length relative to the length of the problem's input. Just as with P and NP, the full extent of PSPACE is not known. PSPACE contains P and NP. It is not known if either of these conclusions are proper. Settling these questions would again be significant steps forward in this theory. The notion of PSPACE-completeness is defined very similarly to NP-completeness, and has been studies alongside the NP-completeness notion. Namely, a problem C is PSPACE complete if it is in PSPACE and if any other PSPACE problem can be reduced to it in polynomial time. As is the case with NP-complete problems, PSPACE-complete problems are quite common and often arise quite naturally. Typical PSPACE-complete problems are or arise from generalized games such as hex or checkers played on boards of unbounded finite size (see [GJ79]). Beyond PSPACE lie the exponential time (EXPTIME) and exponential space complexity classes. A small number of natural problems have been shown complete for these classes (see [GJ79]), and as well EXPTIME is the smallest deterministic class which has been proved to contain NP.

(3) STRUCTURAL COMPLEXITY:

By the early 1970's, the definitions of time and space-bounded complexity classes were precisely established and the import of the class NP and of NP-complete problems realized. At this point effort turned to understanding the relationships between complexity classes and the properties of problems within the principal classes. In particular, attention was focused on NP-complete problems and their properties and on the structure of complexity classes between LOGSPACE and PSPACE. We briefly survey some of these studies here.

(3.1) The Isomorphism Conjecture

In the mid-70's, building on earlier work on G¨odel numberings [HB75, Har82] and in analogy with the well-known result of Myhill from computability theory [Myh55], Berman and Hartmanis [BH77, HB78] formulated their isomorphism conjecture. The conjecture stated that all NP complete sets are P-isomorphic (that is, isomorphic via polynomial time computable and invertible isomorphism). This conjecture served as a springboard for the further study of the structure of NP-complete sets. As evidence for their conjecture, Berman and Hartmanis and others [MY85, KMR87] were able to give simple, easily checkable properties of NP-complete sets which implied they were isomorphic. Using these, they proved that all of the known NP-complete sets were in fact P-isomorphic. This conjecture remains an open question today. A positive resolution of the conjecture would imply that P is not equal to NP. Much effort was focused on proving the converse, that assuming P is not NP then the isomorphism conjecture holds. This remains an open question today. As the number of known NP-complete problems grew during the 1970's, the structure and properties of these problems began to be examined. While very disparate, the NP-complete sets have certain common properties. For example, they are all rather dense sets. Density of a set is measured here simply in the sense of how many string of a given length are in the set. So (assuming a binary encoding of a set) there are 2n different strings of length n. We say that set S is sparse if there is a polynomial p(n) which bounds the number of strings in S of length n, for every n. It is dense otherwise. All known NP-complete sets are dense. One consequence of the isomorphism conjecture is that no NP-complete set can be sparse. As with the isomorphism conjecture, this consequence implies that P is not NP and so it is unlikely that a proof of this consequence will soon be forthcoming. Berman and Hartmanis also conjectured that if P in not equal to NP there are no sparse NP-complete sets. This conjecture was settled affirmatively by the famous result of Mahaney [Mah82]. Mahaney's elegant proof used several new counting techniques and had a lasting impact on work in structural complexity theory.

(3.2) The Polynomial Hierarchy

While numerous hard decision problems have been proved NP-complete, a small number are outside NP and have escaped this classification. An extended classification, the polynomial time hierarchy (PH), was provided by Meyer and Stockmeyer [Sto76]. They defined the hierarchy, a collection of classes between P and PSPACE, in analogy with Kleene's arithmetic hierarchy. The polynomial time hierarchy (PH) consists of an infinite sequence of classes within PSPACE. The bottom (0th) level of the hierarchy is just the class P. The first level is the class NP. The second

level are all problems in NP relative to an NP oracle, etc. Iterating this idea to all finite levels yields the full hierarchy. If P=PSPACE then the whole PH collapses to the class P. However, quite the opposite is believed to be the case, namely that the PH is strict in the sense that each level of the hierarchy is a proper subset of the next level. While every class in the PH is contained in PSPACE, the converse is not true if the hierarchy is strict. In this case, PSPACE contains many problems not in the PH and in fact has a very complex structure (see, for example, [AS89]).

(3.3) Alternation

Another unifying and important thread of results which also originated during the 1970's was the work on alternation initiated out by Kozen, Chandra and Stockmeyer [CKS81]. The idea behind alternation is to classify combinatorial problems using an alternating Turing machine, a generalization of a nondeterministic Turing machine. Intuitively a nondeterministic Turing machine can be thought of as having an existential acceptance criterion. That is, an input to the TM is accepted if there exists a computation path of the machine which results in acceptance. Similarly, we could consider a universal acceptance criterion whereby a nondeterministic machine accepts of all computation paths lead to acceptance. Restricting ourselves to polynomial length alternation, we see that NP can be characterized as those problems accepted by nondeterministic TM running in polynomial time using the existential acceptance criterion. Similarly, the universal acceptance criterion with the same type of machines defines the class co-NP consisting of problems whose complements are in NP. Furthermore, we can iterate these two acceptance methods, for example asking that there exist an path of a TM such that for all paths extending that path there exists an extension of that path which accepts. This idea gives a machine implementation of the notion of alternations of universal and existential quantifiers. It is not hard to see that finitely many alternations results in the finite levels of the polynomial time hierarchy and that alternating polynomial time is the same thing as PSPACE. Other relationship between time and space classes defined using alternation can be found in [CKS81], for example, alternating log space = P and alternating PSPACE = EXPTIME.

(3.4) LOGSPACE

To this point all the complexity classes we have considered contain the class P of polynomial time computable problems. For some interesting problems it is useful to consider classes within P and particularly the seemingly smaller space classes of deterministic log space, denoted L, and nondeterministic log space, denoted NL. These classes provide a measure with which to distinguish between some interesting problems within P, and present interesting issues in their own right. At first glance logarithmic space is a problematic notion at best. An input of length n takes n squares by itself, so how can a computation on such an input take only log n space? The answer lies in changing our computation model slightly to only count the space taken by the computation and not the space of the input. Formally, this is done by considering an "off-line Turing machine." This is a (deterministic or nondeterministic) Turing machine whose input is written on a special read-only input tape. Computation is carried out on read-write work tapes which are initially blank. The space complexity of the computation is then taken to be the

amount of space used on the work tapes. So in particular this space can be less than n, the length of the input to the computation. We define logspace, L, to be the class of languages decided by deterministic Turing machines which use at most O(logn) tape squares. Similarly, NL is defined using nondeterministic Turing machines with the same space bound. It is straightforward to check that L ⊆ NL ⊆ P, and these three classes are thought to be distinct. There are a number of nontrivial problems solvable in L (for example see [LZ77]) as well as problems known to be in NL which are not believed to be in L (for example see [Sav73, Jon75]). Numerous problems in P are thought to lie outside of L or NL. For example, one such problem is the circuit value problem, the problem of determining the value of a Boolean circuit, given inputs to the circuit. The circuit value problem is one of many problems in P which is known to be P complete. These are problems in P which are proved to be complete with respect to log-space bounded reductions, reductions defined analogously to polynomial time bounded reduction in the previous section. Proving a P-complete problem is in L would imply that L = P.

(3.5) Oracles

Oracle results play a unique role in complexity theory. They are meta-mathematical results delineating the limitations of proof techniques and indicating what results might be possible to achieve and which are likely beyond our current reach. Oracle results concern relativized computations. We say that a computation is carried out “relative to an oracle set O” if the computation has access to the answers to membership queries of O. That is, the computation can query the oracle O about whether or not a string x is in O. The computation obtains the answer (in one step) and proceeds with the computation, which may depend on the answer to the oracle query. The first, and still most fundamental oracle results in complexity were carried out by Baker, Gill and Solovay [BGS75]. They proved that there is an oracle reactive to which P=NP and another oracle relative to which P and NP differ. What do these results say about the P vs NP question? They say little about the actual answer to this question. The existence of an oracle making a statement S true is simply a kind of consistency result about S. It says that the statement is true in one particular model or “world” (that is, the oracle set itself). As such, we can conclude that a proof of the negation of S will not itself relativize to any oracle. Thus, as many proof methods do relativize to every oracle, an oracle result provides a limitation to the possible methods used to prove S and hence are evidence that the result is, in this sense, hard. Oracle results have been most useful in delineating theorems which are difficult to prove (i.e., those which do no relativize), from those which might more likely be settled by well-understood, relativizing proof techniques. In particular, the Baker, Gill and Solovay results concerning P and NP question indicate that a proof will be difficult to come by, as has indeed been the case. Since 1978 numerous other oracle results have been proved. Techniques used to achieve these results have become quite sophisticated and strong. For instance, Fenner, Fort now and Kurtz[FFK94] gave a relativized world where the isomorphism conjecture holds where Kurtz, Mahaney and Royer [KMR89] had showed that it fails relative to most oracles. They were the culmination of a long series of partial results addressing this question. There are a few

results in complexity that do not relativize, mostly relating to interactive proof systems (see Section 6.1) but these tend to be the exception and not the rule.

(4) DESCRIPTIVE COMPLEXITY:

Many of the fundamental concepts and methods of complexity theory have their genesis in mathematical logic, and in computability theory in particular. This includes the ideas of reductions, complete problems, hierarchies and logical definability. It is a well-understood principle of mathematical logic that the more complex a problem's logical definition (for example, in terms of quantifier alternation) the more difficult its solvability. Descriptive complexity aims to measure the computational complexity of a problem in terms of the complexity of the logical language needed to define it. As is often the case in complexity theory, the issues here become more subtle and the measure of the logical complexity of a problem more intricate than in computability theory. Descriptive complexity has its beginnings in the research of Jones, Selman, Fagin [JS74, Fag73, Fag74] and others in the early 1970's. More recently descriptive complexity has had significant applications to database theory and to computer-aided verification. The ground breaking theorem of this area is due to Fagin [Fag73]. It provided the first major impetus for the study of descriptive complexity. Fagin's Theorem gives a logical characterization of the class NP. It states that NP is exactly the class of problems definable by existential second order Boolean formulas. This result, and others that follow, show that natural complexity classes have an intrinsic logical complexity. To get a feel for this important idea, consider the NP-complete problem of 3 colorability of a graph. Fagin's theorem says there is a second order existential formula which holds for exactly those graphs which are 3-colorable. This formula can be written as $(\exists A,B,C)(\forall v)[(A(v)\vee B(v)\vee C(v))\wedge(\forall w)(E(v,w) \rightarrow \neg(A(v)\wedge A(w))\wedge\neg(B(v)\wedge B(w))\wedge\neg(C(v)\wedge C(w)))]$. Intuitively this formula states that every vertex is colored by one of three colors A, B, or C and no two adjacent vertices have the same color. A graph, considered as a finite model, satisfies this formula if and only if it is 3-colorable. Fagin's theorem was the first in a long line of results which prove that complexity classes can be given logical characterizations, often very simply and elegantly. Notable among these is the theorem of Immerman and Vardi [Imm82, Var82] which captures the complexity of polynomial time. Their theorem states that the class of problems definable in first order logic with the addition of the least fixed point operator is exactly the complexity class P. Logspace can be characterized along these same lines, but using the transitive closure (TC) operator rather than least fixed point. That is, nondeterministic logspace is the class of problems definable in first order logic with the addition of TC (see Immerman [Imm88]). And if one replaces first order logic with TC with second order logic with TC the result is PSPACE (see Immerman [Imm83]). Other, analogous results in this field go on to characterize various circuit and parallel complexity classes, the polynomial time hierarchy, and other space classes, and even yield results concerning counting classes. The intuition provided by looking at complexity theory in this way has proved insightful and powerful. In fact, one proof of the famous Immerman-Szelepcsenyi Theorem [Imm88, Sze88] (that by Immerman) came from these logical considerations. This

theorem say that any nondeterministic space class which contains logspace is closed under complement. An immediate consequence is that the context sensitive languages are closed under complement, answering a question which had been open for about 25 years. To this point we have considered several of the most fully developed and fundamental areas of complexity theory. We now survey a few of the more central topics in the field dealing with other models of computation and their complexity theory. These include circuit complexity, communication complexity and proof complexity.

REFRENCES:

[ADH97] L. Adleman, J. DeMarrais, and M. Huang. Quantum computability. SIAM Journal on Computing, 26(5):1524–1540, 1997.

[AH87] L. Adleman and M. Huang. Recognizing primes in random polynomial time. In Proceedings of the 19th ACM Symposium on the Theory of Computing, pages 462–469. ACM, New York, 1987.

[Ajt83] M. Ajtai. σ1 1 formulea on finite structures. Journal of Pure and Applied Logic, 24:1–48, 1983.

[AKL+79] R. Aleliunas, R. Karp, R. Lipton, L. Lov´asz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In Proceedings of the 20th IEEE Symposium on Foundations of Computer Science, pages 218–223. IEEE, New York, 1979.

[AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Unpublished manuscript, Indian Institute of Technology Kanpur, 2002.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. Journal of the ACM, 45(3):501–555, May 1998.

[AM77] L. Adleman and K. Manders. Reducibility, randomness, and intractibility. In Proceedings of the 9th ACM Symposium on the Theory of Computing, pages 151–163. ACM, New York, 1977.

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM, 45(5):753–782, September 1998. [AS89] K. Ambos-Spies. On the relative complexity of hard problems for complexity classes without complete problems. TCS, 64:43–61, 1989.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. Journal of the ACM, 45(1):70–122, January 1998.

[ATWZ97] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. SL⊆ L4 3. In Proceedings of the 29th ACM Symposium on the Theory of Computing, pages 230–239. ACM, New York, 1997.

[Bab85] L. Babai. Trading group theory for randomness. In Proceedings of the 17th ACM Symposium on the Theory of Computing, pages 421–429. ACM, New York, 1985.

[BBBV97] C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. SIAM Journal on Computing, 26(5):1510–1523, 1997.

[BBC+98] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. In Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, pages 352–361. IEEE, New York, 1998.

[BCD+89] A.Borodin, S.A.Cook, P.W.Dymond, W.L.Ruzzo, andM.Tompa. Twoapplications of inductive counting for complementaion problems. SIAM J. Computing, 13:559–578, 1989.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity, 1(1):3–40, 1991.

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In Proceedings of the 23rd ACM Symposium on the Theory of Computing, pages 21–31. ACM, New York, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulations unless EXPTIME has publishable proofs. Computational Complexity, 3:307–318, 1993.