International Journal of Scientific Engineering and Research (IJSER)
ISSN (Online): 2347-3878
Impact Factor (2020): 6.733

Analysis and Designs of Algorithms: A Critical Comparison of Different Works on Algorithms

Pranal Dhamdhere[1], Akash Dhamdhere[2], Aishwarya[3]

[1] MCA-DS, School of Engineering, Ajeenkya DY Patil University
Email: *pranaldhamdhere436[at]gmail.com*

[2] MCA-DS, School of Engineering, Ajeenkya DY Patil University
Email: *akashdh7777[at]gmail.com*

[3]Professor, School of Engineering, Ajeenkya DY Patil University
Email: *aishwarya[at]inurture.co.in*

Abstract: ***The research examines the six main textbooks on algorithm design and analysis through an analytical presentation, a critical context, and an integrative conclusion. A set of precise rules for solving a problem, known as an algorithm, can be used to swiftly and precisely produce the intended result from any legitimate input. Algorithms can be viewed as procedural approaches to problems where the focus is on effectiveness and precision. Numerous problem categories include those involving sorting, searching, string processing, graph problems, combinatorial issues, geometrical concerns, and numerical difficulties. Breaking a problem into multiple smaller subproblems of the same kind and nearly the same size, solving each of them recursively, and then merging their solutions to solve the bigger problem is referred to as the "divide-and-conquer" algorithm design technique.***

1. Analytical Exposition

The research examines the six main textbooks on algorithm design and analysis through an analytical presentation, a critical context, and an integrative conclusion. The mind of a computer is an algorithm.

General Science
Simply described, an algorithm is a set of instructions for carrying out a task that are detailed enough for a computer to comprehend. A set of explicit instructions for solving a problem are employed in an algorithm, according to Leviton, A. (2011, p. 3), to produce the desired outcome for any valid input in a finite amount of time. In many cases, algorithms offer procedural answers to issues. In a similar spirit, Mount, D.M. (2003, p.2) states that an algorithm is any well-defined computational process that accepts some values as input and outputs some values as output. The kinds of issues they resolve design principles on which they are founded.

The Stable Matching issue, an algorithmic issue that aptly illustrates many of the themes in Algorithm Design, was introduced in Chapter One of Kleinberg, The book on algorithm design by J. and Taros (2005, p. 1). According to Kleinberg, J., and Taros, E. (2005, p. 4), matching's and perfect matching's naturally occur in modelling a wide range of algorithmic challenges and are discussed frequently throughout the book. The self-enforcing requirement to create a college admissions procedure or a hiring process gave rise to the Stable Matching Problem. The main issue is how to assign candidates to employers so that for every employer E, and every applicant A who is not scheduled to work for E, at for a given set of preferences among employers and applicants.

Kleinberg, J., and Tardos, E. proposed the Gale-Shapley problem (2005, p. 4). The definition of a perfect match is simply a means of pairing the men with the women such that everyone gets married to someone, and nobody gets married to more than one person, preventing both polygamy and singledom. We frequently conduct the studies listed below on algorithms used with any instance of size a:

- The worst-case scenario (the most steps)
- The very bare minimum (best case scenario)
- An average situation (with an average number of steps)
- A series of operations on an input with a size that are over time, an average are called amortisation.

A data structure is a specific method of grouping linked data pieces (Levitin, A., 2011, p. 25). The topic of data architecture is essential for effective algorithmic problem solving since algorithms depend on data to function. The binary tree, the stack, the queue, the graph (via its adjacency matrix or adjacency lists), the array, and the linked list are among the more abstract data structures that can be represented by the most crucial elementary data structures set. An abstract collection of items having a variety of operations that can be applied to them is referred to as an abstract data type (ADT) (Levitin, A., 2011, p. 39). Classes are used in contemporary object-oriented languages to implement ADTs.

The main objective of algorithm design is to find efficient answers to computer problems. According on page 33 of Kleinberg, J., and Tardos, E. (2005), an algorithm runs with a polynomial time, provides qualitatively superior worst-case analytical performance compared to brute-force search, and runs swiftly on actual input cases. A measure of time complexity referred to as time efficiency illustrates how quickly An algorithm executes. complexity, also known as space efficiency, is a measure of how many memory In addition to the space required for its input and output, the method also requires units (Levitin, A., 2011, p. 42). Counting the number of times the procedure is used to identify the time complexity.

The asymptotic order of function growth that represent the effectiveness of algorithms are designated and compared

using the notations O,, and. Without losing generality, it is possible to organise various algorithms' Efficiency can be categorised broadly into the following groups: constant, logarithmic, linear, linearithmic, quadratic, cubic, and exponential (Levitin, A., 2011, p. 95).Although setting up a sum and figuring out the order in which it grows is the primary method for figuring out the number of executions of a nonrecursive algorithm's basic operation, setting up a recurrence relation and figuring out the solution's order of growth is the primary method for figuring out the number of executions of a recursive algorithm's basic operation.

If a function t (n) is bound above by a constant multiple of g(n) for all huge n, that is, if there are certain positive constants c and non-negative integers n0 such that t (n) cg(n) for all n n0, then the expression t (n) O(g(n)) is used (Levitin, A.

If a function t (n) is limited below by a positive constant multiple of g (n) for any large n, or if a positive constant c and a non-negative integer n0 exist such that t (n) cg(n) for all n n0, then the expression t (n) (g(n)) is said to be in (g(n).

If there are some positive constants c1 and c2 and some non-negative integer n0 such that c2g(n) t (n) c1g(n) for every n n0, then a function t (n) is said to be in (g(n) and is denoted t (n) (g(n).

A recursive algorithm's inefficiency may be covered up by its brevity, according to Levitin (2011). When each element is the product of its two immediate predecessors, a series of integers known as the Fibonacci numbers is created. There are numerous methods for calculating the Fibonacci sequencev, many of which have quite varied processing speeds. Algorithm visualisation refers to the use of images to send important algorithmic information (Levitin, A., 2011, p. 95). Algorithm visualisation comes in two flavours: static algorithm visualisation and dynamic algorithm visualisation, also referred to as algorithm animation.

Using brute force to solve a problem is a simple method that frequently relies on the problem statement and the definitions of the concepts at stake (Levitin, A., 2011, p.97). The brute-force approach is recognised for its ease of use and broad applicability, but it has a drawback in the form of its subpar effectiveness. The if after sorting two equal items are still in the same relative position, the technique is stable (Mount, D.M., 2003, p. 8).

The algorithms mentioned below can be used as examples of the brute force method:
Using a definition-based method, perform a sequential search after matrix multiplication and selection. Simple string matching algorithm.

The exhaustive search is a brute-force method for resolving combinatorial puzzles. After choosing those that meet all the conditions, it creates each and every combinatorial item in the problem until the appropriate thing is found and others

The assignment problem, the knapsack problem, and the travelling salesman problem can all be solved using exhaustive-search algorithms (Levitin, A., 2011, p. 130). It has been found that, with a few exceptions, most cases render a comprehensive search impractical. The two main graph-traversal algorithms, depth-first search (DFS) and breadth-first search (BFS), are better alternatives. To put it into the form of is the simplest way to study a number of important aspects of a graph.

A general algorithm design method known as "decrease-and-conquer" makes use of the connection between a remedy for one particular manifestation of a problem and a fix for a different, more minor instance of the same issue. To further benefit from the relationship, top down (usually in recursive fashion) tactics can be employed (Levitin, A., 2011, p.167). The three primary decrease-and-conquer versions are as follows:

- reduce-by-a-constant, most frequently by one (for example, insertion sort)
- drop by a constant factor, typically by a factor of two (binary search, for example) (For instance, Euclid's algorithm) variable-size-decrease

The decrease-(by one)-and-conquer technique is easily applied to the sorting problem using insertion sort (Levitin, A., 2011, p.167). Insertion sort is a (n2) method both in the worst case and the average case, while the average case is almost twice as fast. On almost sorted arrays, the approach works well.

A graph with orientations along its edges is referred to as a digraph (Levitin, A., 2011, p.168). The goal of the topological sorting problem is to arrange a digraph's vertices so that each edge starts at a different vertex than the vertex it points to. If and only if a digraph is a directed acyclic graph, or does not contain directed cycles, then this problem has a solution. The topological sorting problem can be resolved using either of two approaches: a direct use of the decrease-by-one strategy or a depth-first search approach. It makes sense to employ the minimal-change algorithms in the decrease-by-one method.

Create algorithms for generating fundamental combinatorial things.

When searching in a sorted array, the Binary search strategy, which is based on a decrease-by-a-constant-factor algorithm, is especially effective (Levitin, A., 2011, p.168). A balance scale can be used to detect fake currency, and other examples include the Josephus problem, Russian peasant multiplication, exponentiation by squaring, and the Josephus issue. The size reduction varies from one algorithm iteration to the next for several decrease-and-conquer algorithms. Examples of such variable-size decreasing methods include Euclid's method, the partition-based solution to the selection problem, interpolation search, and searching and insertion in a binary search tree.

A broad algorithm design strategy known as "divide-and-conquer" splits a larger problem into a number of smaller sub-problems of the same sort, solves each one iteratively, and then combines the solutions to arrive at the answer to the larger problem (Levitin, A., 2011, p. 198). This method serves as the foundation for many efficient algorithms despite its limitations and inadequacy to more obvious

automated answers. Several divide-and-conquer algorithms have running times T (n) that satisfy the recurrence T (n) = aT (n/b) + f (n). The sequence of the solutions to the Master Theorem will develop is predetermined. Divide and conquer is a well-known three-step strategy that is employed by both Mergesort and Quicksort, according to Erickson, J. (2019, p. 29).

Organise the provided instance of the problem into a number of separate, smaller cases that are precisely the same.

Send the Recursion Fairy each smaller instance. Combine the smaller instance solutions to get the complete instance solution.

A divide-and-conquer sorting algorithm called merge sort sorts an input array by splitting it into two equal halves, sorting them repeatedly, and then combining the two sorted halves to sort the original array (Levitin, A., 2011, p.198). The number of key comparisons is extremely near to the theoretical minimum, and the procedure is always in (n log n) time. The main disadvantage is the large additional storage space needed. The divide-and-conquer algorithm known as MergeSort operates recursively. The array is split into two roughly equal-sized sub-arrays, which are then sorted iteratively before being combined n times (Mount, D.M., 2003, p.9). Merge sort is a dependable sorting algorithm. The only one is the MergeSort algorithm, though.

Make two sub-arrays from the input array, both roughly the same size. Every sub-array is mergesorted iteratively. Combine the newly sorted sub-arrays to create a single sorted array.

The efficient heap data structure, This is a priority queue data structure implementation, is the foundation of the heap sort algorithm. a component with a modest key value is eliminated as a result of the priority queue's support for key insertion operations. The creation of a heap for n keys and retrieval of the minimal key can both be done in (log n) time and (n) time, respectively (Mount, D.M., 2003, p.9). Despite being an in-place sorting method, HeapSort is unstable A heap's k smallest values can be removed in n + k log n time, according to Mount (D.M., 2003, p.9). A heap is advantageous in situations when an element's priority varies since each priority change (key value) may be handled by Quicksort, a divide-and-conquer sorting algorithm that divides incoming elements based on how important they are in relation to a chosen element (Levitin, 2011, p.168). As one of the bestQuicksort is a well-known technique for sorting randomly ordered arrays because it has a quadratic worst-case efficiency. According to Mount, D.M. (2003, p. 8), Quicksort divides the array into components that are lower and higher than the pivot after first choosing a random "pivot value" from it. Then, Quicksort recursively sorts each component. On contemporary processors, QuickSort is frequently cited as the quickest rapid sorting algorithm. Quicksort's inner loop compares each element to a single pivot value that is easily accessible and may be stored in a register. Two elements in the array are compared using the other algorithms. Quicksort is an in-place sorting algorithm that doesn't utilise any other array storage, but Mount, D.M. As not stable (p. 8). The following are the crucial processes for Quicksort, according to Erickson, J. (2019, p. 27):

Choose a key element from the list.the pivot element, the pivot element's smaller and larger counterparts, and the pivot element are separated into three sub-arrays.

Recursively quick sort the first and last sub-arrays.

The traditional binary tree traversals (preorder, inorder, and post order) and similar algorithms, which call for recursive processing of both left and right sub-trees, are an example of the divide-and-conquer strategy (Levitin, A., 2011, p.168). Whole a given tree's empty subtrees can be replaced with one-of-a-kind external nodes.

The divide-and-conquer method requires approximately n1.585 one-digit multiplications to multiply two n-digit numbers. By employing the divide-and-conquer technique, Strassen's approach, which generally requires two 2 2 matrices must be multiplied by seven times, can multiply two n n matrices with about n2.807 multiplications. According to Levitin, A. (2011, p.198), the convex-hull problem and the closest-pair issue are two key computational geometry issues that the divide-and-conquer technique can be used to.

A heap is a binary tree in which one key is assigned to each of its nodes, provided that the two conditions below are satisfied (Levitin, A., 2011, p. 227):

Simply put, the binary tree is full (shape characteristic).

The key of each node must be greater than or equal to the keys of its offspring, according to the parental dominance or heap attribute.

The fourth broad algorithm design (and issue-solving) strategy addressed by Levitin, A. (2011, p. 250), is known as transform-and-conquer and refers to a set of methods based on the idea of transforming a problem into one that is simpler to solve.

The transform and-conquer approach comes in three main versions, which are:

Through instance simplification, a problem is changed from one instance to another with a particular a feature that makes it easier to tackle the problem. Effective examples of this technique include rotations in AVL trees, list presorting, and Gaussian elimination.

The term "representation change" refers to the process of switching from one instance of a problem representation to another. Using a 2-3 tree, heaps and heapsort, as well as Horner's rule for evaluating polynomials, and two binary exponentiation algorithms are examples of this category.

Problem reduction is the process of transforming a given problem into a different problem that can be addressed by a well-known method. Reductions to linear programming and reductions to graphing issues are two examples of this technique.

A heap, according to Levitin (2011), p. 250, is a binary tree with keys (one per node) that satisfy the parental dominance requirement and is essentially complete. The efficient implementation of priority queues depends on heaps, which are binary trees that are frequently implemented as arrays. Heaps also form the basis of heapsort. Theoretically significant sorting procedure known as "heapsort" entails storing array members in a heap and then repeatedly eliminating the largest element from the heap that is left. The approach is in-place and executes in O(n log n) in both the worst case and average case (Levitin, A., 2011, p. 250). AVL trees are binary search trees that preserve the balance to the extent that it is possible for a binary tree to do so.

By transforming systems of linear equations into analogous systems with upper-triangular coefficient matrices that are straightforward to solve using back substitutions, the process known as "Gaussian elimination" may be utilised to solve systems of linear equations (Levitin, A., 2011, p. 251). Gaussian elimination requires about [n3 /3] multiplications. the only

The best method for polynomial evaluation without coefficient preprocessing is Horner's rule, which requires n multiplications and n additions to evaluate an n-degree polynomial at a specific position. The synthetic division method is one of Horner's rule's many advantageous side effects.

Optimising a linear function of several variables that is subject to restrictions in the form of linear equations and linear inequalities is the goal of linear programming. If the variables are not needed to be integers, there are efficient methods that can solve very large instances of this issue with thousands of variables and restrictions. The latter are a far more challenging set of issues and are known as integer linear programming.

Critical Context

A linear programme, in general, entails the maximisation or minimization of a linear function of some variables while taking into account linear restrictions on those variables. One of the most prevalent issues that can be resolved in polynomial time is linear programming. Numerous optimisation issues can be solved in polynomial time by immediately converting them into polynomial-size linear programmes (Khuller, S., 2012, p.83). The simplex algorithm, invented by Dantzig, is the most popular technique for solving linear programmes. By introducing slack variables, the approach first transforms the linear inequalities into equality constraints. Combinatorial optimisation issues can be solved using the Primal-Dual Method (Khuller, S., p.138).

Optimisation concerns are frequently dynamic programming issues, according to Mount, D.M., 2003, p.

(Assuming various limitations, determine the least expensive or most expensive solution). The strategy separates a larger issue in a way comparable to divide-and-conquer. Into easier issues that are then routinely solved. Due to dynamic programming problems' slightly different nature, divide-and-conquer tactics are frequently useless. In general, dynamic programming can be used to solve optimisation problems if the original problem can be divided into smaller subproblems and the recursion among the subproblems has the condition of optimum substructure.

This implies that the best answers to the sub-problems can be added to get the best answer to the main issue. A dynamic programming technique typically enumerates all conceivable division methods, in contrast to the conventional divide-and-conquer strategy. the fundamentals of structure within the is breaking the problem down into smaller (and ideally simpler) sub-problems and describing the answer to the larger problem in terms of solutions to the smaller problems.

A table-based layout where the solutions to the subproblems are kept at a table because they are frequently reused. Bottom-up computing combines the answers to smaller subproblems with those to bigger subproblems to solve them.

According to Erickson, J. (2019, p. 190), a (simple) graph is formally defined as a pair of sets (V, E), where V is a freely chosen non-empty finite set and E is a set of pairs of V's elements, which we refer to as edges. According to Mount, D.M. (2003, p. 30), a graph is a group of nodes (sometimes called vertices) connected by a group of edges. For many application issues, graphs offer a very versatile mathematical model. The vertices or nodes of a directed graph (or digraph) G = (V,E) and the edges of G are made up of a finite set V and E, respectively. E can be thought of as a simple binary relation on V (Mount, D.M., 2003, p.30). the collection E containing distinct, unordered pairs of vertices that make up a graph that is not directed G = (V,E) is made up of the edges and a limited collection V of vertices. The Digraph and Graph, respectively, are shown in Figure 1 below.

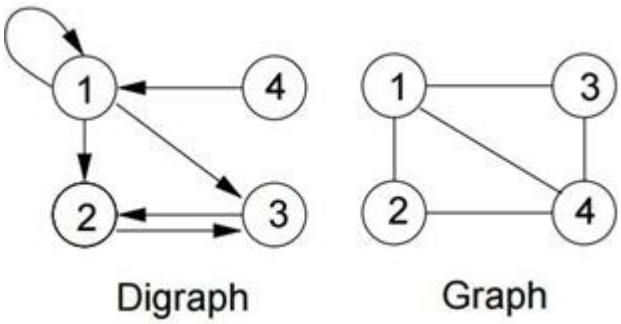

Figure 1: Digraph and Graph

Splay trees are an effective way to show the strength of amortised analysis because they work as search trees without requiring any explicit balancing criteria (Khuller, S., 2012, p. 7). A splay operation takes O(log n) time to amortise (Khuller, S., 2012, p. 10).

A planar embedding is a mapping of a graph's vertices and edges to the plane where no two edges intersect. A graph is referred to as being planar if it has a planar embedding (Khuller, S., 2012, p.58). Another way to think of planar graphs is as graphs that are planarly embedded on a globe. A planar graph is helpful as they are used in the VLSI design

context. K5 and K3,3 are the smallest non-planar graphs (Khuller, S., 2012, p. 62).

A programme that completes It is considered to have polynomial time if it completes in a period of time O(nk), where k is an independent constant for n. If a problem can be solved by an algorithm in polynomial time, that problem is said to be polynomially solvable (Mount, NP-completeness: We can state that issue Q is NP-complete if we can show both (a) Q N P and (b) X N P X Q.p. 75; S. Khuller, 2012).

We define the following problem classes in this context:

P: All decision-making tasks that can be finished in a polynomial amount of time are included in this set. These tasks are typically referred to as "easy" or "efficiently solvable" tasks (Mount, D.M., 203, p.58).

P is a subset of NP, which is the set of all decision problems that can be verified in polynomial time (Mount, D.M., 2003, p. 59). There are many There are NP class issues that are considered to be quite simple, but there are also some that are very difficult.NP stands for "nondeterministic polynomial time," not "not polynomial."

This issue is NP-hard. If we could resolve this issueWe would be able to in polynomial time resolve all NP-hard issues as well (Mount, D.M., 2003, p. 59). It should be noted that an issue need not belong to the class NP in order to be NP hard. However, it is generally accepted that no NP-hard problem can be solved in polynomial time because it is widely held that all NP problems cannot be solved in that amount of time.

If a problem is both (1) in NP and (2) NP-hard, it is said to be NP-complete. NPC is therefore NPNP-hard.

The most frequent method utilised while creating algorithms is reduction. Writing an algorithm for X that employs an algorithm for Y as a black box or subroutine is referred to as reducing one problem X to another problem Y (Erickson, J., 2019, p.921). Importantly, the functionality of the resulting algorithm for Y cannot in any way influence how correct the method for X is. According to J. Erickson (2019, p. 22), when we create algorithms, we might not be fully aware of If it is finished in time O(nk), where k is a constant that is independent of n, then it is said to have polynomial time. If there is a formula that applied as a foundation to address even more complex issues. Recursion is a very potent type of reduction, and Erickson, J. (2019, p. 22) suggests that it can be loosely characterised as follows:

Solve the problem directly if it can be done for the particular case that has been given.

If not, simplify it into one or more variants of the original issue.

Integrative Conclusion

A process known as planar embedding assures that no two edges cross by mapping a graph's vertices and edges to the plane. Apparently, S.

A graph is deemed to be planar if it has a planar embedding, according to Khuller (2012), on page 58. Another way to think of planar graphs is as graphs that are planarly embedded in a sphere. Planar graphs are useful for VLSI design because they are applicable. The smallest non-planar graphs, according to S. Khuller (2012), are K5 and K3,3.

Any algorithm with a polynomial time specification completes in O(nk) time, where k is an independent variable from n. If a Using a polynomial time algorithm resolve a problem, it is regarded as having been resolved in polynomial time. (Mount, Divide-and-conquer is a general algorithm design strategy that entails breaking a larger problem into several smaller sub-problems that are all roughly the same size and type, solving each of them recursively, and adding the solutions to the solution of the initial problem (Levitin, A., 2011, p. 198).Although many efficient algorithms are built using this approach, it can occasionally be ineffective and inferior to more straightforward algorithmic solutions. The mergesort divide-and-conquer sorting technique is used to divide an input array.information about publishing.

Acknowledgement

We appreciate Professor Aishwarya's important advice and assistance during the research process. His knowledge

These insights were crucial in determining the emphasis and direction of our research. We are also appreciative of Ajeenkya DY Patil University's MCA Data Science programme for giving us the tools and assistance we required to finish this research.

References

[1] Algorithms, J. ERICKSON, 2019, paperback, Creative Commons, ISBN 978-1-792-64483-2, Copyright 2019. Available at http://jeffe.cs.illinois.edu/teaching/algorithms/ is Jeff Erickson's work under a Creative Commons Attribution 4.0 International Licence.
[2] E. Taros and J. Kleinberg (2005). Algorithm Design, 2006. Paper, ISBN 0-321-29535-8, Pearson Education, Inc.
[3] (2012) S. KHULLER. University of Maryland's Department of Computer Science course notes from January 26, 2012, "Design and Analysis of Algorithms"
[4] LEVITIN, A. Third Edition, 2015, Pearson, New York, ISBN-13: 978-0-13-231681-1, ISBN-10: 0-13-231681-1. a description of algorithm creation and research.
[5] Mount, D.M. (2003). Design and Analysis of Computer Algorithms, Department of Computer Science, University of Maryland, Fall 2003.
[6] M.A. WEISS (2014). Florida International University, Fourth Edition, Data Structures and Algorithm Analysis in C++, ISBN-13: 978-0-13-284737-7 (all. paper), ISBN-10: 0-13284737-X (all. paper), QA76.73.C153W46 2014.