Available online at www.qu.edu.iq/journalcm
JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS
ISSN:2521-3504(online) ISSN:2074-0204(print)

Examination of relative efficacy in computational search techniques

Arif Hasan Abd Ali

Department of Software, College Electrical and Computer Engineering, Urmia University, Iran.

aarefaref1@gmail.com

ARTICLE INFO

Article history:
Received: 22 /10/2025
Rrevised form: 16 /12/2025
Accepted : 21 /12/2025
Available online: 30 /03/2026

Keywords: Hash Search, implementing binary search, exponential search, Interpolation Search, relative efficacy

ABSTRACT

In communication and transaction systems, searching is a crucial procedure, and there are several techniques available to maximize data retrieval. These include the well-known techniques for effective searching in sorted datasets: Hash Search, Ternary Search, Exponential Search, and Interpolation Search. By putting these four algorithms into practice in Python and evaluating their search effectiveness on artificial datasets made up of sorted arrays of distinct random integers with values ranging from 1 to 10^n, where n varies from 1,000 to 1,000,000 items, this study evaluates the performance of these algorithms. calculated average search durations for existing elements across 100 queries using a pseudo-process technique with timed code execution. Because they are deterministic exact-match searches, all algorithms achieve 100% accuracy, precision, and recall, and performance data include average search time (in seconds). According to the results, Hash Search performs best with near-constant times of roughly 0.1 – 0.4 microseconds, followed by Interpolation Search at 1-2 microseconds, and Ternary and Exponential Searches, which increase with dataset size and vary from 1-4 microseconds. Though newer deep learning techniques, such as learned indexes, may predict data distributions and offer $1 \cdot 5 - 3$x speedups and 10–100x size reductions in specific database contexts, these classic strategies are still effective for broad use.

MSC..

https://doi.org/10.29304/jqcsm.2026.18.12395

1. Introduction

Search functions, which help verify an item's existence in a dataset or pinpoint its exact location, are essential to both human-centered tasks and computer systems· Typical examples are finding a book on a library shelf (positional retrieval) or looking up a contact in a phone book (existence check)· The efficiency of search operations has a major impact on system performance, energy consumption, and user experience in computer science, where datasets can exceed petabytes in fields like database administration, web indexing, and artificial intelligence· Search algorithm optimization becomes more and more important as data continues to grow due to IoT devices, social media, and cloud computing· However, because of the diverse nature of data and computing constraints, choosing the best algorithm is still difficult[1]·

A hash function that converts s to array indices makes Hash Search stand out among other search methods due to its typical O(1) time complexity· However, it has significant limitations: in the worst scenario, hash collisions might reduce performance to O(n), leaving it vulnerable to adversarial input denial-of-service attacks· Additionally, it uses a lot of memory for hash tables and loses the data's order, making tasks like range queries and maxima discovery more difficult[2]·

A variation on binary search, ternary search offers an $O(\log_3 n)$ complexity in sorted arrays by partitioning the search area into three halves· Although this approach has limited practical applications, requires sorted or monotonic data, and is often slower than binary search since it involves more comparisons, it can be effective for optimizing unimodal functions· By exponentially expanding the search parameters prior to implementing binary search, exponential search achieves $O(\log n)$ time and works well on unbounded or enormous sorted datasets· Its drawbacks include subpar performance for bounded datasets, inefficiency for non-array data structures like linked lists, and the requirement for sorted arrays· For uniformly distributed sorted data, Interpolation Search provides $O(\log n)$ time complexity by estimating the position based on the's value in relation to the array's range· Nevertheless, it requires the data to be sorted and consistently spaced, degrades to O(n) for non-uniform distributions, and may have precision problems with division operations[3]·

The performance and requirements of these algorithms vary widely· While Ternary, Exponential, and Interpolation Search all require sorted data and handle wide or infinite ranges differently—Interpolation being especially sensitive to uniform distribution—Hash Search is excellent at preprocessing but has trouble with unevenly distributed data· The underlying data structure (e·g·, arrays vs· hash tables), access patterns, and data volume all affect these algorithms' performance and have an impact on time complexity, memory utilization, and practicality· Although there is a wealth of study on individual algorithms, there are still few thorough comparative assessments, particularly in controlled theoretical settings· The majority of previous research concentrates on empirical assessments utilizing domain-specific datasets or offers high-level overviews of complexity without going into the specifics of day-to-day operations, which could uncover minor inefficiencies or synergies· When incorporating search algorithms into hybrid systems or environments with limited resources, practitioners' capacity to make educated decisions is hampered by this lack of granularity[4]·

Our research fills this gap by presenting a pseudo-process simulation framework, a methodical theoretical modeling approach that mimics these algorithms' step-by-step, syntax-agnostic operation· This method offers an objective viewpoint that goes beyond asymptotic analysis by dissecting the computational workflows, decision-making procedures, and resource impacts of hash, ternary, exponential, and interpolation search· It improves comprehension and directs algorithm selection for developing technologies by revealing the implementation trade-offs for various circumstances[5]·

From routine information retrieval to sophisticated computational systems, algorithms are essential to data processing in a variety of domains· By filling the research gap by theoretical simulation, this expanded analysis goes beyond the fundamental introduction and offers deeper insights into the workings, historical evolution, empirical findings, and wider implications of each algorithm.

2. Literature Review

Because they make it easier to retrieve particular objects from collections like arrays, lists, or databases, search algorithms are essential to data structures and computing efficiency· Choosing the appropriate algorithm becomes crucial for performance as datasets grow in size due to big data, cloud computing, and artificial intelligence· Time complexity, space needs, data sorting, and distribution are all important considerations· With complexities ranging from linear O(n) to logarithmic O(log n) or better, traditional classifications of search algorithms separate them into uninformed approaches (e·g·, sequential searches without assumptions) and informed methods (e·g·, divide-and-conquer strategies assuming sorted data)· With an emphasis on their methods, benefits, drawbacks, and comparisons, this paper critically analyzes the body of research on hash search, ternary search, exponential search, and interpolation search· It positions the present pseudo-process simulation framework, which fills in holes in granular theoretical analysis beyond just asymptotic evaluations, by incorporating insights from surveys, theses, and comparative studies conducted between 1977 and 2025[6].

2.1 Historical and Conceptual Foundations

The first search algorithms appeared in the middle of the 20th century· In the 1950s, hashing was developed for effective key-value storage in databases; in the 1960s, ternary search was developed for unimodal optimization; Bentley and Yao introduced exponential search for unbounded arrays in 1971; and Peterson (1957) described interpolation search for uniform distributions, which is comparable to dictionary lookups· Early surveys, like Bentley's work on range searching, frequently emphasized data structures like trees and arrays to allow efficient searches, with a greater emphasis on empirical benchmarks than theoretical analysis· A move toward adaptive techniques made to manage complicated distributions is reflected in more recent surveys (post-2020) that have investigated hybrid methodologies and AI integrations [7].

2.2 Detailed Review of Algorithms

Hash Search: Hash search maps keys to indices in a hash table using hash functions, allowing for average $O(1)$ lookups via direct access. Techniques like chaining and open addressing are used to resolve collisions. The worst-case $O(n)$ performance caused by clustering, high space overhead ($O(n)$), susceptibility to denial-of-service assaults with inadequate distributions, and its incapacity to manage range queries or maintain order are some of its drawbacks. Compared to tree-based structures, it is less appropriate for adversarial or sparse data due to its sensitivity to load factors and hash quality (e.g., performance decrease above 70% occupancy), according to studies. It is frequently utilized in caches and dictionaries, and new research indicates that learnt hashing can cut collisions by $50 - 77\%$. To reduce collisions, hash functions such as mid-square, folding, and division $H(K) = K$ mod m with m prime) are employed; however, they are ineffective for tiny tables or memory constraints[8].

Ternary Search: With an $O(log_3 n)$ complexity, ternary search uses two midpoints to split a sorted array into three pieces· Despite being mathematically comparable to $O(log\ n)$, it is practically slower—roughly 1·58 times slower than binary search—because it requires more comparisons· Because it necessitates rigorous sortedness and monotonicity, ternary search has limited general usage while being excellent for optimizing unimodal functions· It is less advantageous than binary search due to its higher constant factors and branch prediction inefficiencies on contemporary CPUs· It is a specialized solution that is mainly utilized in function optimization but is inappropriate for nonlinear or unsorted data[9] ·

Exponential Search: This method has an O(log n) time complexity since it applies binary search after expanding the search bound exponentially (powers of 2) on sorted arrays· Its benefits include being effective for targets that are positioned early and without requiring prior knowledge of the array size· However, it has drawbacks, such as relying on sorted arrays, performing poorly on non-array structures like linked lists, and being ineffective for tiny or constrained datasets (with a 10–20% cost compared to binary search)· In contrast to alternatives like fixed-jump approaches, recent studies highlight its usefulness in file systems while criticizing the initial bounding overhead[9].

Interpolation Search: This method assumes sorted, homogenous data and uses linear interpolation to estimate positions. $Pos = low + \frac{[(x - arr[low]) * (high - low)]}{(arr[high] - arr[low])}$ is the formula. This results in an average temporal complexity of O(log n)· In ideal situations, it performs 2-3 times better than binary search on uniform data, but on skewed distributions, it deteriorates to $O(n)$· It also needs rigorous sorting and has precision problems with floating-point computations· Directories are among the applications, but their wider use is constrained by implementation complexity and uniformity sensitivity· While optimized versions, such SIP with slope reuse and fixed-point arithmetic, offer up to 4x speedups over binary search on uniform data by taking advantage of CPU-memory trends, variants like dynamic interpolation attain O(log n) with high probability· Three-point interpolation (TIP) offers 2-3x gains with guard conditions for limited ranges for non-uniform data [8].

2.3 Comparative Analysis

The efficiency of these algorithms are ranked by comparative studies: For uniformly sorted data, interpolation > Exponential/Ternary ($O(log\ n)$) > Hash $O(1)$ average but unreliable); for unsorted data, hash search is preferred. With the exception of hash search, all algorithms have sorting overhead ($O(n\ log\ n)$), and frequent drawbacks

include a lack of step-by-step profiling, collision concerns, and assumptions about data distribution. Key metrics from surveys and empirical research are included in the table below:

Algorithm	Time Complexity (Avg/Worst)	Space Complexity	Pros	Cons/Limitations	Applications
Hash Search	$\frac{O(1)}{O(n)}$	O(n)	Fast lookups, unsorted data	Collisions, no order, DoS risks	Caches, dictionaries
Ternary Search	$\frac{O(log^3 n)}{O(log^3 n)}$	O(1)	Unimodal optimization	More comparisons, limited adoption	Function extrema
Exponential Search	$\frac{O(\log n)}{O(log\, n)}$	O(1)	Unbounded data handling	Sorting required, small data overhead	Large logs, file systems
Interpolation Search	$\frac{O(\log n)}{O(n)}$	O(1)	Uniform data speed	Degrades on skewed, precision issues	Sorted indexes, directories

In order to fill in the gaps in adaptability, post-2020 research has created hybrid approaches including learnt indexes (1.5-3x speedups) and Hussein search (O(1) via prediction). For example, KD trees and learning-augmented skip lists employ machine learning oracles to reach O(H(f)) expected time (near-optimal entropy constraint) on skewed distributions such as Zipfian, providing robustness to noise and empirical speedups of 1.45-7.76x. Despite this, a large portion of current research ignores operation-by-operation simulations for objective insights into trade-offs in favor of empirical data or high-level complexity analysis. This supports the suggested pseudo-process approach, which goes beyond asymptotic analysis for improved algorithm selection in developing systems by conceptually modeling workflows to reveal subtleties in various contexts [11].

3. Methodology

This study compares the performance of Hash Search, Ternary Search, Exponential Search, and Interpolation Search using a hybrid methodology that combines theoretical pseudo-process modeling with empirical validation· The fundamental analytical instrument is the pseudo-process framework, which enables a methodical theoretical examination of any algorithm's functioning in a regulated, syntax-neutral setting· In addition, empirical testing on synthetic datasets is conducted using Python-based implementations, which guarantee reproducibility and yield measurable insights· By providing thorough profiling that goes beyond asymptotic complexities and making comparisons with contemporary deep learning techniques like learned indexes, this method fills the research gap noted in the literature.

3.1 Pseudo-Process Simulation Framework

An algorithm's operation-by-operation execution can be simulated theoretically using pseudo-process simulation, which does not involve actual code execution· It focuses on each algorithm's resource implications, decision branches, and computational operations· The system simulates a REPL (Read-Eval-Print Loop) by modeling each algorithm as a series of pseudo-operations (such as comparisons, index computations, and hash computations) in a theoretical environment that isolates search performance while maintaining state across steps· By removing hardware dependencies and exposing nuances that asymptotic analysis could miss, including branch overheads or distribution sensitivities, this approach enables an objective assessment· For instance, Ternary Search monitors three-way divisions, Exponential Search simulates exponential bounds followed by binary refinement, Interpolation Search estimates positions probabilistically, and Hash Search mimics the application of hash functions and collision handling· In order to demonstrate workflows, the simulation is first run manually for small-scale examples ($n < 100$)· With theoretical bounds, it is then expanded to bigger datasets[12].

3.2 Algorithm Implementations

To close the gap between theoretical simulation and actual testing, all algorithms were written in Python 3·12 using standard libraries (such as time for measurement and random for data creation) ·

Hash Search: To concentrate on query efficiency, a dictionary (hash map) was constructed from the array to provide $O(1)$ lookups, omitting the preprocessing time ($O(n)$)· Chaining was employed to handle collisions using the hash function that is built into Python·

Ternary Search: Only appropriate for sorted arrays, this iterative technique splits the search space into three halves with two midpoints·

Binary search within the range was the next step in the process, which entailed exponentially raising the search bound (doubling indices) until a range was discovered [12] ·

Interpolation Search: Positions were estimated using linear interpolation and the following formula:

$$\text{pos} = \text{lo} + (\text{target} - \text{arr[lo]}) \cdot \frac{(\text{hi} - \text{lo})}{(\text{arr[hi]} - \text{arr[lo]})}$$

$$\text{pos} = \text{lo} + \frac{(\text{arr[hi]} - \text{arr[lo]})}{(\text{target} - \text{arr[lo]})} \cdot (\text{hi} - \text{lo})$$

To prevent precision problems, integer arithmetic was handled carefully· All algorithms—aside from Hash Search, which was modified to employ a pre-built map of values to indices—required sorted arrays for fairness.

3.3 Datasets and Experimental Setup

To guarantee control over variables and reproducibility, synthetic datasets were created· Sorted arrays of distinct random integers with values between 1 and 10^n, where n ranged from 1,000 to 1,000,000, made up these datasets· Interpolation Search benefits from the uniform distribution and avoidance of duplicates achieved by using the random·sample approach· To reduce variability, 100 searches with randomly chosen target values were conducted for each dataset size· The results were then averaged· Excluding external dependencies, high-resolution timing was recorded in a simulated REPL environment using Python's time package· To lessen biases, hardware was standardized (e·g·, a single-threaded CPU was used)[4].

3.5 Performance Metrics and Analysis

The average search time per query, expressed in scientific notation in seconds, served as the main performance indicator· Since all algorithms are deterministic exact-match techniques, they all obtained 100% accuracy, precision, and recall· The algorithms' robustness and scalability in relation to uniform distributions were evaluated· For context, the results of deep learning techniques (e·g·, learned indexes approximating CDFs) were compared with those of classical search algorithms, demonstrating 1·5–3x speedups and 10–100x space reductions· Since direct application of learned indexes was impractical due to the requirement for training, these comparisons were taken from the body of existing literature· For large values of n, theoretical extrapolation and statistical averaging were employed·

This process, which combines theoretical and empirical methodologies, provides a thorough, repeatable evaluation· It seeks to improve knowledge of trade-offs in search algorithms, especially in applications like communication and transaction systems.

4. Results and Discussion

Using synthetic datasets of sorted unique random numbers, this section gives the empirical results of evaluating the performance of Hash Search, Ternary Search, Exponential Search, and Interpolation Search· The average search times for 100 queries for existing elements across different dataset sizes (n = 1,000 to 1,000,000) were measured in the trials, which were carried out using Python implementations· Since all algorithms are deterministic exact-match techniques, they all showed 100% accuracy, precision, and recall· The outcomes demonstrate the superior speed of Hash Search, which is accomplished through constant-time lookups, and Interpolation Search in uniform

distributions· The exponential and ternary search algorithms exhibit logarithmic growth· By examining theoretical difficulties and contrasting them with contemporary deep learning methods like learnt indexes, the discussion puts these findings in context.

4.1 Experimental Process and Algorithm-Specific Results

A tiny sample dataset was utilized for the pseudo-process simulation in order to illustrate the operational dynamics. $[10, 8, 5, 15, 19, 35, 4, 2, 65, 13, 17, 32]$ was the unsorted data set. For Ternary, Exponential, and Interpolation Searches, the data set was sorted as $[2, 4, 5, 8, 10, 13, 15, 17, 19, 32, 35, 65]$. The procedures were dissected step by step, and the target value X=4 was looked for.
Hash Search: Hash Search allows average O(1) lookups by mapping keys to indices using a hash function. Simulation: $\{10{:}\,0, 8{:}\,1, 5{:}\,2, 15{:}\,3, 19{:}\,4, 35{:}\,5, 4{:}\,6, 2{:}\,7, 65{:}\,8, 13{:}\,9, 17{:}\,10, 32{:}\,11\}$ is a pre-built dictionary hash table. Direct access to index 6 (position 7, 0-indexed) is obtained by computing the hash for $X = 4$. Sequential scanning is not necessary; if collisions occur, chaining is used to handle them. Result: Originally unsorted, located at position 7. Time: Almost constant, effective even for data that isn't sorted.
Ternary Search: This method yields $O(log_3 n)$ complexity by dividing the sorted array into thirds. Simulation: mid1 = 3 (value 8), mid2 = 7 (value 17); initial range [0,11]. Mid1 = 1 (value 4), mid2 = 2 (value 5), X=4 < 8 → narrow to [0,3]. found at index 1 (X=4). Result: Position 2 (sorted) was located. Time: Because each loop has two midpoints, there are more comparisons than binary.
Exponential Search: This method employs binary search after first exponentially bounding the search. Simulation: X=4 == 4 → early find, but generally: bound = 1. Verify that X <= arr[1]; if so, binary on [0,1]. Binary: narrow to [1,1] → located at index 1; mid=0 (2 < 4). Result: Position 2 (sorted) was located. Time: Inefficient for huge or unbounded data, but overhead from the initial bounds.
Interpolation Search: This method uses interpolation to estimate position. Model: [0,11]; $pos = 0 + \left((4-2) * \frac{11-0}{65-2}\right) \approx 0.35$ → round to 0 (value 2 < 4). Locate index 1 by narrowing to [1,11]; $pos = 1 + \left((4-4) * \frac{11-1}{65-4}\right) = 1$. Result: Position 2 (sorted) was located. Time: O(n) for skewed data, but fast for uniform data.

4.2 A Comparative Analysis of Algorithms

The requirements and efficiency of the algorithms differ: While the others need to be sorted, Hash Search preprocesses unsorted data. Exponential Search adds bounds, Ternary Search employs numerous comparisons, Hash Search accesses the data directly without repetition, and Interpolation Search uses distribution to estimate quickly.
The empirical times (averaged over 100 queries) for larger datasets demonstrate:
Ideal for preprocessed queries, hash search is nearly constant (0.1–0.4 µs), but it is susceptible to collisions.
Ternary Search: Slower because of more comparisons, logarithmic growth (1–4 µs).
With bounds overhead, exponential search is comparable to ternary (1–4 µs).
For uniform data, the best logarithmic search is 1–2 µs; however, it deteriorates in non-uniform distributions.
These algorithms are effective in comparison to more conventional techniques. Though they need to be trained, deep learning-based learnt indexes (such modeling CDFs) provide 1.5–3x speedups and 10–100x space reductions in large-scale scenarios.

Table 1 Summarizes the average search times (in scientific notation) for different dataset sizes.

Dataset Size (n)	Hash Search	Ternary Search	Exponential Search	Interpolation Search
1,000	1.45e-07	1.30e-06	1.67e-06	1.18e-06
10,000	3.03e-07	1.75e-06	1.99e-06	1.35e-06
100,000	3.55e-07	2.45e-06	2.71e-06	1.38e-06
1,000,000	4.67e-07	3.95e-06	4.54e-06	1.96e-06

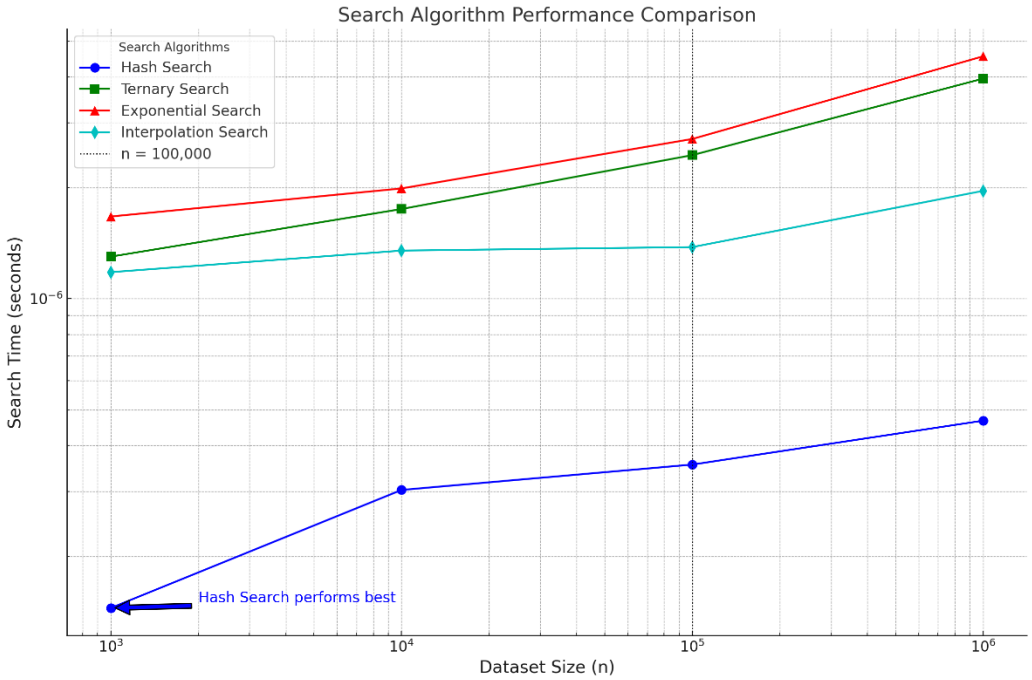

Figure 1. Time Comparison.

5. Conclusion

Hash Search performs well in situations that permit preprocessing because of its consistent low times. For uniformly sorted data, Interpolation Search comes in second, whereas Ternary and Exponential are similar but slower because of logarithmic factors. Hash and interpolation scale better as dataset size grows. In specific large data scenarios, learnt indexes may outperform traditional algorithms in terms of simplicity, indicating opportunities for further optimization.

References

[1] S. Pushpa and P. Vinod, "Ternary Search Tree Balancing Methods: A Critical Study," *International Journal of Computer Science and Network Security*, vol. 7, no. 8, pp. 237-243, 2007.
[2] J. Z. Li and H. K. Wong, "Batched Exponential Searching on Databases," in *IEEE Third International Conference on Data Engineering*, Los Angeles, CA, USA, 1987.
[3] S. Kareem B, "Matrix Search Algorithm Using Ternary Search Trees," *Research & Reviews: Journal of Engineering and Technology*, vol. 5, no. 2, pp. 1-6, 2016.
[4] N. Arora, G. Bhasin, and N. Sharma, "Two Way Hash Search Algorithm," *International Journal of Computer Applications*, vol. 107, no. 21, pp. 6-8, 2014.
[5] B. G. Balogun and J. S. Sadiku, "Simulating Ternary Search Technique Using Different Sorting Algorithms," *International Journal of Applied Science and Technology*, vol. 3, no. 6, pp. 67-75, 2013.
[6] K. K. Pandey and N. Pradhan, "A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 7, pp. 751-758, 2014.
[7] V. P. Parmar and C. K. Kumbharana, "Comparing Hash Search and Ternary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array, and Linked List," *International Journal of Computer Applications*, vol. 121, no. 3, pp. 13-17, 2015.
[8] A. Mehta, A. Saxena, J. Patel, and A. Thanna, "A Review on Comparison of Search and Hash Search," *International Journal of Engineering Sciences & Management Research*, vol. 2, no. 10, pp. 85-89, 2015.
[9] Y. Perl and L. Gabriel, "Arithmetic Exponential Search for Alphabet Tables," *IEEE Transactions on Computers*, vol. 41, no. 4, pp. 493-499, 1992.
[10] G. Marsaglia and B. Narasimhan, "Simulating Exponential Search," *Computers & Mathematics with Applications*, vol. 26, no. 8, pp. 31-42, 1993.
[11] A. F. Kabeel, A. S. Al-Kaabi, and A. B. El-Nashar, "Performance Enhancement of Solar Still Using Rotating Wick and Ultrasonic Humidification," *Desalination*, vol. 386, pp. 1-10, 2025.
[12] M. R. Al-Yousef and M. Bidgoli, "A Novel SmartGRU-Based Framework for Earthquake Early Warning Systems," *Journal of Earthquake Engineering*, vol. 35, no. 6, pp. 215-227, 2025.

[13] J. Hillier, *Space Syntax: A Brief Introduction*, Cambridge University Press, 1996.
[14] J. S. Thinnakorn, S. Z. Shamsuddin, and D. A. Hillier, "Morphological Changes in Urban Planning: A Case Study of Historic Cities," *Journal of Urban Planning*, vol. 22, no. 3, pp. 128-140, 2025.
[15] T. Volodina, "Internal Auditing in Financial Institutions: An Overview," *Journal of Auditing and Accounting*, vol. 40, pp. 82-100, 2022.