Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526

Volume 14 Issue 02, 2025

Linked-Arrays: A Mathematical Approach

[1]Sarveshkumar G. Nasare, [2]Sunil M. Wanjari

[1,2] Department of Computer Science and Engineering, St. Vincent Pallotti College of Engineering and Technology, Nagpur, Maharashtra, India

Email: [1]nasaresk@gmail.com

[1]0009-0009-4210-8689

[2]0000-0003-0337-4115

Peer Review Information

Submission: 05 Nov 2025

Revision: 25 Nov 2025

Acceptance: 17 Dec 2025

Keywords

Linked-arrays, spiral-traversal algorithm, hybrid data structure, dynamic array, binary search.

Abstract

Arrays and linked-lists serve as fundamental data structures, each with distinct advantages and limitations. Arrays, with their contiguous memory allocation, offer efficient data access. Similarly, linked-lists, connected via pointers, allow dynamic insertion and deletion. Both are linear data structures and have specific advantages as well as limitations such as, binary search cannot be applied on linked-list. There also exists a hybrid data structure in which, data field of node is replaced with array but still binary search in not applicable. In this work, a hybrid data structure of array & linked-list called as "Linked-Arrays" has been proposed along with an algorithm called "Spiral-Traversal Algorithm", which allows applying binary search on Linked-Arrays. The performance of Linked-Arrays varies between performances of its parent data structures & hence, possess most of the advantages of both.

Introduction

Arrays offer a compact way to store collections of the same data type. Elements reside in consecutive memory locations, enabling efficient access through a base address and an index offset. However, efficiency comes at a cost: arrays have a fixed size set during declaration. This static allocation ensures predictability but hinders flexible insertions and deletions. These operations require shifting elements, resulting in higher time complexity for such operations.

Similar to arrays, a Linked-List is also a linear data structure. Unlike arrays, Linked-Lists store data in non-contiguous memory locations. Each element (node) holds data and a pointer to the next node, forming a chain. This allows for dynamic sizing and efficient insertions/deletions, but random access is slow and searching methods like binary search are inapplicable. Linked-Lists also use additional space for the pointers.

This research explores "Linked-Arrays," a hybrid data structure combining arrays and linked-lists (Fig. 1, 2). It stores entire arrays within linked-list nodes, offering advantages over existing hybrids like "Unrolled Linked-Lists" [1]. Linked-Arrays achieve faster search due to array caching and require less memory for pointers compared to standard linked-lists. This translates to relatively quicker insertion, deletion, and traversal operations.

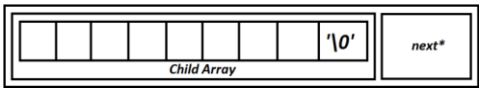

Figure 1: Node in Linked Arrays

In this paper, an algorithm called "Spiral-Traversal Algorithm" has been proposed. By using this algorithm, faster searching algorithms like **Binary Search** can be applied on the Linked-Arrays data structure. In this Algorithm, arrays

stuck together using linked nodes are simulated as a single array where element access at index ***i*** follows a two-step process:

1. Locate the "*parent node position*" (***n***) in the linked list that holds the child array containing the target element.
2. Within the located child array, determine the "*virtual index*" (***i'***), which represents the element's position relative to the child array's indexing.

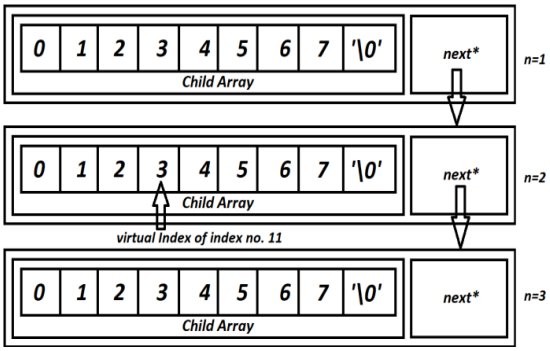

Figure 2: Linked-Arrays

For instance, to access element at index 11, we need to access *virtual index* (***i'***) 3, where *parent node position (**n**)* is 2.

Literature Survey

The [1] is an online article, which explains about the *Unrolled Linked-List* data structure. The *Unrolled Linked-List* is a data structure Similar to the *Linked Arrays* data structure mentioned within this paper, whereas it's working and data storage method is different than that of the *Linked-Arrays*.

Sara et al. (2020) [2], proposes the Hybrid Array List (HAL), a novel data structure that addresses limitations of traditional arrays in dynamic resizing. HAL combines the structure of a linked-list with fixed-size sub-arrays within each node. The size of these sub-arrays is determined by a user-defined initial value. As elements are added or removed, HAL adjusts its size by manipulating both the number of nodes in the linked-list and the elements within the sub-arrays. This approach offers potential performance benefits over standard linked-lists and arrays, particularly in scenarios involving frequent insertions and deletions.

The paper in [3] explores the efficiency of *linked-lists* and *arrays* in data structures. It highlights the superior space and time complexity of *linked-lists*, especially when dealing with large datasets. The paper also notes that *linked-arrays* require significantly less space than *arrays* and *linked-lists*, making them an efficient choice for data storage and retrieval. This study contributes to the discourse on optimizing data structures for better performance.

The authors at [4], provides a comparative analysis of the *linear search* and *binary search* algorithms. The research investigates the performance of these fundamental search techniques when applied to linear lists implemented using three common data structures: static-*arrays*, *dynamic-arrays*, and *linked-lists*. Their work evaluates the time complexity and practical efficiency of each algorithm-data structure combination.

Methodology

To implement the *Spiral-Traversal Algorithm*, the size of *arrays* (denoted by **len**) in all of the nodes must be same. The Read (Traversal/Searching), Update (Write/Insert) and Delete are the important operations to be done on any dataset. To support the Insert Operation on *Linked-Arrays*, new Node is required to be added at the end of *Linked-Arrays*. The shifting of elements will have to be done considering it as a single bigger *array*. The blocks remained empty during Insert operation are called as a "*Vacancy*" (refer fig. 3) and lies in the last node of linked-arrays. The first of those empty block stores the character '**\0**' denoting the end.

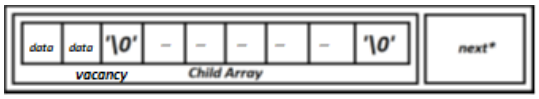

Figure 3: end Node containing vacancies

The following are the steps used to locate an *index* (**i**) using *Spiral-Traversal Algorithm*:-

Find the *parent node position* (**n**) of the *linked-array* which contains data of index **i** using the below formula:-

$$n = (i + 1) / len \quad \textbf{...(1)}$$

(If this formula returns Floating-Point value (for example, **2.18**) then the immediate greater integer should be used (i.e, **3**) as a value of **n**.)

Once parent node is located, find the *virtual-index* (**i`**) using the below formula:-

$$i` = i - ((n - 1) \times len) \quad \textbf{...(2)}$$

Now, traverse from head node to the **nth** (parent) node using *pointer* to the next *node* and then access *virtual index* (**i`**) of the parent node, which contains exact data which should be on index (**i**) if there was a single *array*.

The following are the steps used to Insert a new element into *Linked-Arrays*:

1. Find the *parent node position* (**n**) (using formula in equation 1) which contains index (**i**) where new element is to be inserted.
2. Insert new Node (array length = **len**) at the end of *Linked-Arrays*.
3. Shift the elements as done in traditional *array* to create space for storing the element at desired

index and insert the element in there using respective *virtual index* (**i`**).
4. The *vacancies* will be situated in the last node, insert '**\0'** into the first *vacancy*, denoting the end of the *array*.
The following are the steps used to Delete an existing element from the *Linked-Arrays*:
1. Find the *parent node position* (**n**) (using the same first formula) which contains index where delete operation is to be performed.
2. Delete the element at that index using respective *virtual index* (**i`**), shift the elements (including '**\0**' of first *vacancy*) as done in traditional *array* to cover-up empty memory space.
3. If all blocks in the child *array* of last *node* becomes a *vacancy*, then delete that last *node*.

Results
The *Spiral-Traversal Algorithm* allows to access required index (**i**) in *Linked-Arrays* efficiently. Since the indices of *Linked-arrays* are calculated in constant time **O(1)**, *Binary Search* can be successfully applied on *Linked-Arrays*. The time complexity of *Spiral-Traversal Algorithm* is **O(1)** for best-case and **O(total no. of nodes in *linked-Arrays*)** for worst-case. The Time Complexity of Binary search on *Linked-Arrays* is, best-case: **O(1)** and worst-case: $\mathbf{O(m\log_2 n)}$ (where, **n** = **m*len** & **m** = total no. of nodes in Linked Arrays). Due to some extra steps and calculations required to access the *index* (**i**), *Spiral-Traversal Algorithm* takes more time than accessing *index* (**i**) in a traditional *array*. However, the time is very much less as compared to the time required to access an element in *Linked-List*.
Since, *Linked-arrays* are nothing but, the linearly connected *Nodes*, Insert and delete operation become efficient just like *Linked-list*, if whole *Node* is inserted or deleted. Whereas, the time complexity of single element insert/delete operation in *Linked-Arrays* takes constant time in best case & for worst case it takes **O(total no. of nodes * len)** time.
The Insert and Delete operation performed on the *Linked-Arrays* take the same amount of time as that of an array. For both operations, **O(1)** is for best case and worst case is of **O(m)** (**m** = total no of elements in whole *Linked-Arrays*).

Discussions
The Linked-Arrays and The Spiral-Traversal Algorithm together enables multiple smaller linearly connected arrays to be simulated as a single bigger array and hence, the use of binary search on the Linked-Arrays is efficiently possible. This availability is very useful because Linked-Arrays can achieve theoretically unlimited length, whereas traditional arrays have fixed lengths declared at the time of definition. It should be noted that the memory spaces gets wasted (vacancies) while adding new Node during Insert operation as that of in Linked-list.

Acknowledgment
We thank all of the anonymous readers and reviewers for spending their precious time, calmly reading this paper. I hope this research work is worthwhile to you and your valuable time.
Thank you!

References
https://www.geeksforgeeks.org/unrolled-linked-list-set-1-introduction

Sara MR, Klaib MF, Hasan M. Hybrid Array List: An Efficient Dynamic Array with Linked List Structure. Indonesia Journal on Computing (Indo-JC). 2020;5(3):47-62.

Lokeshwar B, Zaid MM, Naveen S, Venkatesh J, Sravya L. Analysis of time and space complexity of array, linked list and linked array (hybrid) in linear search operation. In 2022 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI) 2022 Dec 8 (Vol. 1, pp. 1-6). IEEE.

Parmar VP, Kumbharana CK. Comparing linear search and binary search algorithms to search an element from a linear list implemented through static array, dynamic array and linked list. International Journal of Computer Applications. 2015 Jan 1;121(3).

Karlsson M, Dahlgren F, Stenstrom P. A prefetching technique for irregular accesses to linked data structures. InProceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550) 2000 Jan 8 (pp. 206-217). IEEE.

Samanta D. Classic data structures. Terminology. 2001;2:1.

Puntambekar AA. Data structures. Technical Publications; 2020 Dec 1.

Subero A. Linear Data Structures. InCodeless Data Structures and Algorithms: Learn DSA Without Writing a Single Line of Code 2020 Feb 14 (pp. 19-29). Berkeley, CA: Apress.

Hopcroft JE, Ullman JD, Aho AV. Data structures and algorithms. Boston, MA, USA:: Addison-wesley; 1983.

Mrena M, Varga M, Kvassay M. Experimental Comparison of Array-based and Linked-based List Implementations. In2022 IEEE 16th International Scientific Conference on Informatics (Informatics) 2022 Nov 23 (pp. 231-238). IEEE.

Srivastava K. Array-Linked Data Structure: Introducing a Hybrid Model of Memory Management and Faster and Easier Insertion and Reallocation Procedures. International Journal of Data Structure Studies. 2023;1(1):1-1p.