IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

Analysis of Arraylist and Linked list

K. Renuka Devi

Department of Computer Science, Dr.Mahalingam College of Engineering and Technology, Tamilnadu, India

Corresponding Author: krenukagiri@gmail.com

DOI: https://doi.org/10.26438/ijcse/v7i5.15661570 | Available online at: www.ijcseonline.org

Accepted: 10/May/2019, Published: 31/May/2019

***Abstract*—** In the concept of data structures, the List plays a major role in the allocation of data. "A list in java is an interface that can extend to collection interface". A list can be implemented in two ways: Array list and Linked list. Array list is a class which provides growable array of list ADT. Linked list provides different implementation of the List ADT. There are different kinds of linked lists support in data structures which could be singly linked list, doubly linked list and circularly linked list. This paper deals with the analysis of array list and linked list (i.e.) singly linked list by performing operations such as insertion, deletion, searching and provides results based on time complexity to decide which would be better and efficient for allocation of data.

***Keywords*—**list, arraylist, linkedlist, data structure.

I. INTRODUCTION

The term data structure is one of the predominant ways for allocating and organizing the data. There are different types of data structures which would be Array, List, Stack, Queue, Trees and graph [1].

This paper deals with list data structure. List could be implemented in two ways. They are array list and linked list. Array list is a list data structure which is implemented using dynamic arrays. So, that it could be able to grow as needed. The linked list is also a list data structure that is implemented as a node with two entities: the data and a pointer. The data which holds the actual value and the pointer holds the address of the next value. This paper deals with linked list implementation which is based on singly linked list.

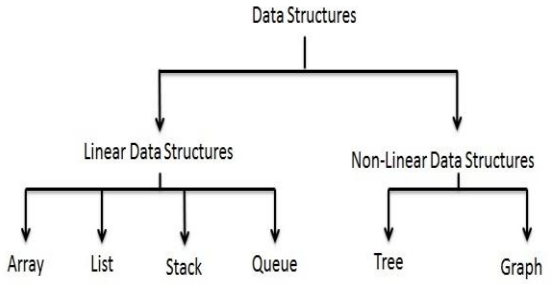

Figure 1. Data structures – Types

This paper is organized as follows. Section I contain the Introduction of the Arraylist and linked list. Section II contains the Implementation of Array and Linked lists. Section III contains the Analysis part of two lists described. Section IV contains the results and brief discussions of the methodology analysed. And finally the Section V contains the conclusion and future scope.

II. IMPLEMENTATION

A. Array list:

Array list is a class which extends abstract list and implements the list interface. It can grow as needed because its supports dynamic arrays. Standard arrays which are implemented in java are of fixed length. It cannot grow (or) shrink, this becomes huge disadvantage that the user needs to know the size of the array in-advance in order to implement and to get the values be allocated.

So, we tend to use the Array list, it is created with initial size but when this size gets exceeds (or) when the element gets removed the array gets grow in the former type and the array gets shrinks in the later type [2,10].

Constructors for creating Array list:

Arraylist() – builds an empty list.

Arraylist(constructor) – builds an array list which gets initialized with the constructor.

Methods:

There are many methods to initialize and to create array list. They are,

Void add(int index, object element) – Inserts the specified element at the specified index.

© 2019, IJCSE All Rights Reserved

Boolean add(object object_name) – It appends the specified element to the end of the list.

Array list Creation:

ArrayList<string> al = new ArrayList<>();

This is used to create a new arraylist with the name specified as "al". Using this name we can able to perform several operations such as to add an element, to delete an element and to search for an element [3].

al.add("her") - to add an element "her".
al.remove("her") – to remove an element "her".

```
public static void main(String[] args)
{

  ArrayList<String> ll = new ArrayList<>();
  ll.add("Ravi");
  ll.add("rani");
  ll.add(0,"renu");
  ll.remove("Ravi");
 System.out.println(ll);
```

Figure 2. Array list

B. Linked list:

A linked list in data structure is implemented as a node which contains two entities such as data and a pointer. It is in the form of sequence of links which is interconnected. Each node contains the data and the pointer holds the address of next data in a sequence and the last node contains the address as the null pointer [4,10].

There are different kinds of linked lists. They are,

- Singly linked list – links only in forward direction.
- Doubly linked list – links both in forward and backward direction.
- Circularly linked list – links in circular form.

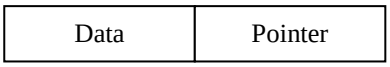

Figure 3. Node representation

B.1. Singly linked list

The navigation of an item which is in the forward direction. Each node contains the reference to an element in the sequence and contains the address of the next element but it doesn't contain any reference to the previous node.

To store this list only the reference to the 1st node must be stored and the last node points to null.

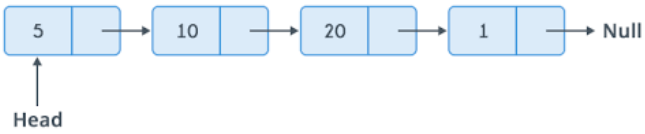

Figure 4. Singly linked list

Creation of Node:

```
Node head;                  //head of list
static class Node           //defines as static so that main() can access it
{
int data;                   //like structure in c
Node next;
Node(int d)                 //constructor
{
data = d;
next = null;
}
}
```

The node can be created by initializing "Node head" and then the constructor has been created by the above process.

Linked list creation:

```
public static Node insert(singlylinkedlist list,int data)
{
    c++;
    Node new_node = new Node(data); //new node with data
    new_node.next = null;
    Node last = null;
    if(list.head == null) //if the list is empty
        list.head = new_node; // insert list with new node
    else
    {
        last = list.head;  //if not empty create new node ca
    while(last.next != null)   //check if there is value in
    {
        last = last.next;    // head moves to nextnode if th



    }

    last.next = new_node; // insert as last node

    }
    System.out.println("i"+c);
return last;

}
```

Figure 5. Linked list

The figure 5 shows that how to insert the element in the list using singly linked list.If the list is empty,it creates new node and then inserts the first data.If the list is not empty,the list traverse until last node and then creates new node at last[5].

© 2019, IJCSE All Rights Reserved

III. ANALYSIS

A. *Array list-time complexity:*

Table 1. Array list – Time complexity(operations)

	Beginning (ms)	Middle (ms)	End (ms)
Insertion	300	340	400
Deletion	150	170	210
Search	100	125	150

The table 1 shows the time complexity of arraylist for the operations such as insertion, deletion and search where the data has been inserted, deleted and searched at the beginning, middle and at the end.

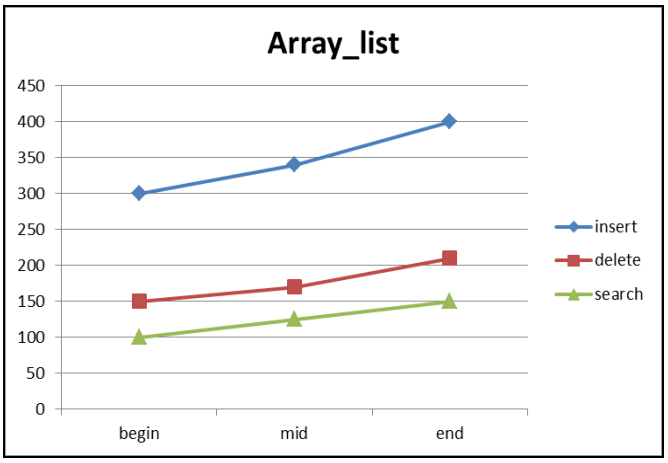

Figure 6. Array list-Time complexity

The Figure 6 shows the time complexity of Arraylist for the operations such as insertion, deletion and search. From the above graph we infer that the element inserted in the beginning takes less time than at the end [7]. This is because when inserting the new element the array needs to grow in size and also it doesn't utilize pointer.

B. *Linked List – Time Complexity*

Table 2. Linked list – Time Complexity(operations)

	Beginning (ms)	Middle (ms)	End (ms)
Insertion	100	139	200
Deletion	95	100	132
Search	282	310	390

The table 2 shows the time complexity of linked list for the operations such as insertion, deletion and search where the data has been inserted, deleted and searched at the beginning, middle and at the end.

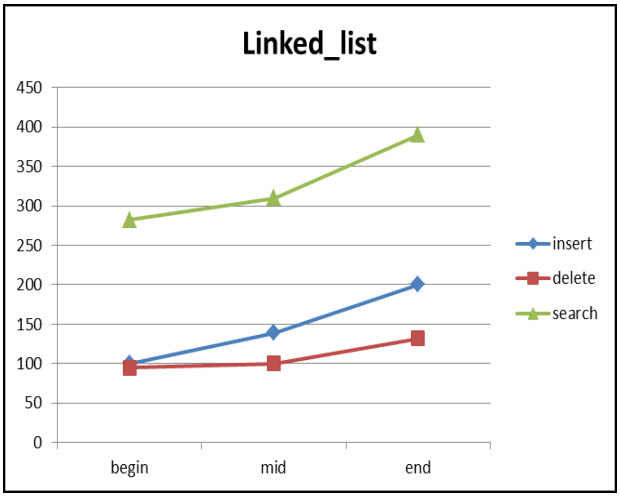

Figure 7. Linked list – Time Complexity

The Figure 6 shows the time complexity of Linked list for the operations such as insertion, deletion and search. From the above graph we infer that the element inserted in the beginning takes less time than at the end [8]. The insertion and deletion takes more or less same time because it makes use of pointer by means of creation of node.

C. *Comparison – Arraylist vs. Linked list*

Table 3. Comparison – Arraylist vs. Linked list(operations).

	Beginning (ms)	Middle (ms)	End (ms)
Insert-array	300	340	400
Insert-linked	100	139	200
Delete-array	150	170	210
Delete-linked	95	100	132
Search-array	100	125	150
Search-linked	282	310	390

The table 3 shows the time complexity for the comparison of arraylist and linked list for the operations such as insertion, deletion and search where the data has been inserted, deleted and searched at the beginning, middle and at the end.

© 2019, IJCSE All Rights Reserved

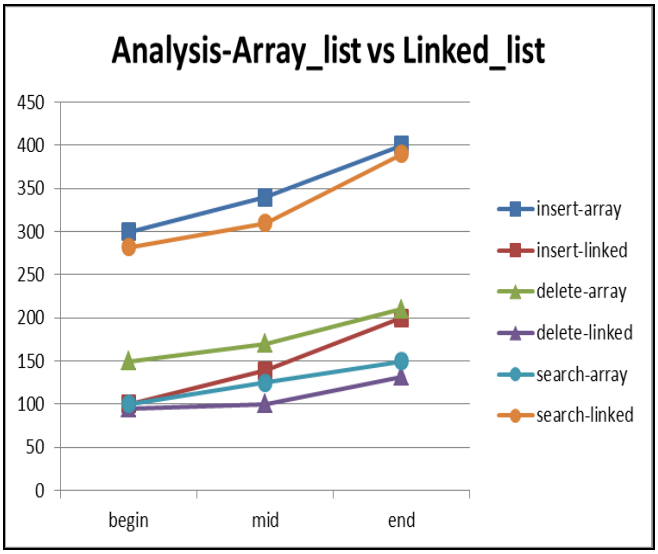

Figure 8. Comparison – Arraylist vs. Linkedlist

The Figure 8 shows the comparison of arraylist and linked list. From the graph we infer that insertion and deletion of linked list takes less time than arraylist. This is because the linked list makes use of links. So, the new node gets inserted only at the last by moving the pointer [9]. Whereas, the arraylist has to increase the size of the array. The deletion in linked list deletes only the node specified by deleting the address and replacing the pointer. Whereas, the arraylist has to shrink the size of the array after deleting the element. The search operation of linked list takes more time than arraylist. This is because the linked list has to seek from initial node till the final node if the element present at the last. Whereas, the arraylist has to search only at the specified index [10].

IV. Results And Discussion

The insertion, deletion and search operations have been performed for arraylist and linked list (i.e.) singly linked list. When analysing the above two methods, we get the result as the insertion and deletion operation of linked list consumes less time than arraylist. But the search operation in linked list consumes more time than arraylist. This is because the linked list utilize pointer to access the element whereas the arraylist uses dynamic array and it has to grow or shrink the size of the array. From the above analysis we infer that Linked list is more efficient and better than Arraylist for allocation of elements.

Table 4. Time Complexity

	Arraylist	Linked list(singly linked list)
Insertion	O(n)	O(1)
Deletion	O(n)	O(1)
Search	O(1)	O(n)

V. Conclusion and Future Scope

List in data Structure plays a vital role in the allocation of data because it utilizes node to access the data. It uses pointer to move across the node for insertion as well as deletion operation. This paper mainly focuses on which method would be efficient for data allocation while comparing arraylist and linked list. Linked list takes only O(1) for insertion whereas the array list takes O(n). As a result it is better to utilize linked list to allocate the data more efficiently and to delete the data. Aside from comparing only with singly linked list, the other kinds of linked lists and operations can be taken into consideration in future.

References

[1] Stelios Xinogalos, Maya Satratzemi, *"An analysis of students' difficulties with ArrayList object collections and proposals for supporting the learning process"*, In the Proceedings of the 2008 IEEE International Conference on Advanced Learning Technologies, Cantabria, **Spain**, pp. **180-182**, **2008.**

[2] Gaifang Dong, Xueliang Fu, "*An Improved Pathfinding Algorithm Based on Sorted Linked List and Indexed Array*", IEEE Transaction, Vol.**6,** Issue.**4**, pp.**978-981**, **2008**.

[3] Anshu Yadav, Aruna Bhat, Rajni Jindal, "*Stack implementation of adjacency list for representation of graphs*", In the Proceedings of the 2008 IEEE International Conference on Advanced Learning Technologies, Noida, Uttar Pradesh, India, pp. **213-216**, **2013.**

[4] Shruti rishab panday, "*A Heuristic Approach of Sorting Using Linked List*", In the Proceedings of the IEEE Second International Conference on Computing Methodologies and Communication, Erode,India, pp. **446-450, 2018**.

[5] Karuna, Garina Gupta, "*Dynamic Implementation Using Linked List", International Journal of Engineering Research & Management Technology*", Vol.**1,** Issue.**5**, pp.**44-48**, **2014**.

[6] H. C Thomas, E. L Charles, L. R Ronald, and S. Clitlord, "*Introduction to Algorithms*", Second Edition, MIT Press and 609 McGraw- Hill, ISBN 0-262-03293- 7. Section 1.1 Algorithms, pp.**5, 2001**.

[7] W.H. Butt, and M. Y. Javed, "*A New Relative Sort Algorithm based on mean value*". IEEE Conference on Multi topic, **2008**.

[8] Devareddi Ravi Babu, R Shiva Shankar, V Pradeep Kumar, Chinta Someswara Rao, D Madhu Babu, V Chandra Sekhar, "*Array-Indexed Sorting Algorithm for natural numbers*", IEEE, pp. **606-609,2011**.

[9] Wong, J., Vernon, A. Field, J., "*Evaluation of a Path-Finding Algorithm for Interconnected Local Area Networks*", Selected Areas in Communications, pp. **1463-1470, 1987**.

© 2019, IJCSE All Rights Reserved

[10] Nagendra Singh, "*Role of Suffix Array in String Matching: A Comparative Analysis*", International Journal of Computer Sciences and Engineering, Vol. **3,** Issue.**6**, pp.**89-93**, **2015**.
[11] Sourabh Shastri, "*A GUI Based Run-Time Analysis of Sorting Algorithms and their Comparative Study*", International Journal of Computer Sciences and Engineering, Vol. **5,** Issue.**11**, pp.**217-221**, **2017**.
[12] A.Chitra, P.T. Rajan, "*Data Structures*", second edition, **2007**.

Authors Profile

K.Renuka devi completed her Bachelor of Engineering in Computer Science from Dr.Mahalingam College of Engineering and Technology, Tamilnadu in 2018.She is currently pursuing Master of Engineering in Computer Science from Dr.Mahalingam College of Engineering and Technology, Tamilnadu. She is a member of ACM since 2018. Her main research work focuses on Security in Computing, Data Structures, and Web Mining.

© 2019, IJCSE All Rights Reserved