A New Improved Circular Skip List

1. Upinder Kaur, Research Scholar, Dept. of Computer Science and Applications, KUK.
2. Dr. Pushpa Rani Suri, Professor, Dept. of Computer Science and Applications, KUK.

Abstract: Skip list is a data structure with an ordered sequence of elements. It consist of layer of linked list. They consist of a layered structure and all nodes are in the bottom layer. These nodes are reduced to half towards upper layers and thus a pyramid-like structure is formed, which facilitates search, insertion and removal operations. A circular linked list is a type of linked list in which the last node of the list points back to the first node. In this paper we proposed a new data structure improved circular skip list (ICSL). ICSL is created with the help of circular linked list and skip list data structures. In circular linked list, operations are performed on a single round robin list. However, our new data structure consists of circular link lists formed in layers which are linked in a conical way with improved priority search feature. Time complexity of search, insertion and deletion equals to O (log N) in an N-element improved circular skip list data structure. Improved circular skip list data structure is employed more effectively (O(log N)) in circumstances where circular linked lists (O(N)) are used with improved priority searching technique.

1. Introduction

Various disciplines in computer sciences benefit from data structures directly or indirectly. Different data structures are used as solutions to various problems. New data structures are sometimes required due to the limitations such as processing, time and hardware or inefficiency of current data structures. Sometimes, a data structure is preferred over another one because of its processing speed. New data structures emerged because dynamic and static structures are required [1]. Taking these factors into consideration, it is evident that new data structures and algorithms will continue to emerge [2]. A circular linked list is a type of linked list in which the last node of the list points back to the first node. In single or double linked list the last node contains a NULL pointer since there is no next node, whereas in a circular linked list the "next" pointer of the last node contains the address of the first node (Figure 1). Therefore it is called a circular linked list. A circular linked list has a "start" node, but no "end" node. In a Circular Linked List all the nodes are linked in continuous circle (Figure1).

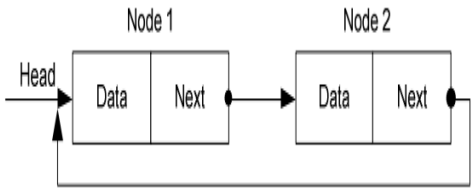

Figure 1. Circular Linked list structure

It can be both singly or doubly linked list. In a circular linked list elements can be added to the back of the list and removed from the front in constant time. Both types of

circularly -linked lists benefit from the ability to traverse the full list beginning at any given node. This avoids the necessity of storing first node and last node, but we need a special representation for the empty list, such as a last node variable which points to some node in the list or is null if it's empty. This representation significantly simplifies adding and removing nodes with a nonempty list, but empty lists are then a special case. Circular linked lists are most useful for describing naturally circular structures, and have the advantage of being able to traverse the list starting at any point. They also allow quick access to the first and last records through a single pointer [3].

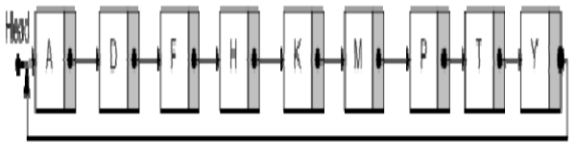

Figure 2. Circular Linked List

2. Skip List

Skip list data structure, which was introduced by Pugh [4,5,6] is a data structure alternative to binary tree search structure. Linked lists are used in skip list data structure and it is aimed to facilitate searching, insertion and deletion through placing elements in a pyramid-like order at different levels. In this data structure, elements are placed at different levels randomly. First, all nodes are placed at level 0 and, starting from left row and skipping each 2ith node (i=0,..,MaxLevel (15)), pointers representing each level are created towards the top. The list at level 0 is the linked list at the bottom in skip list data structure and encompasses all nodes. Each list from bottom to the top is arranged as an index of the previous list [1,7]. Search, insertion and delete algorithms of nodes in skip list data structure is discussed in article written by Pugh [4,5]. In addition, several studies have been conducted so far on the improvement and analysis of skip list data structure algorithms. These studies are about level optimization in skip list data structure [1], effects of P threshold values in creation of random level and to the performance of skip list data structure [2], a simple optimistic skip list algorithm [8], analysis of an optimized search algorithm for skip lists [9], skip lists and probabilistic analysis of algorithms [10], deterministic skip lists [11], concurrent maintenance of skip lists [5]. Various data structures and algorithms were also created apart from skip list data structure such as skip graphs [12], tiara (peer-to-peer network maintenance algorithm) [13] and corona [14]. Time complexity is O(N) for search, insertion and deletion processes when linked and ordered lists are used. On the other hand, the time complexity in which these processes are performed is O(log N) in skip list data structure [6]. In a search algorithm, a node is searched from upper levels to lower levels. During insertion, first, the node to be inserted is searched. If not found, new value is inserted to the matching location starting from a random level and pointers and lists are updated. The process is repeated for other levels where a node is to be inserted. Search is performed from the top level to lower levels for removal operations. The node is deleted when found and pointers and lists are updated. The process is repeated other levels where the node is available [2].

A group of data consisting of {A,C,F,H,K,M,P,T,V,X,Z} elements as a skip list is shown in Figure 3.

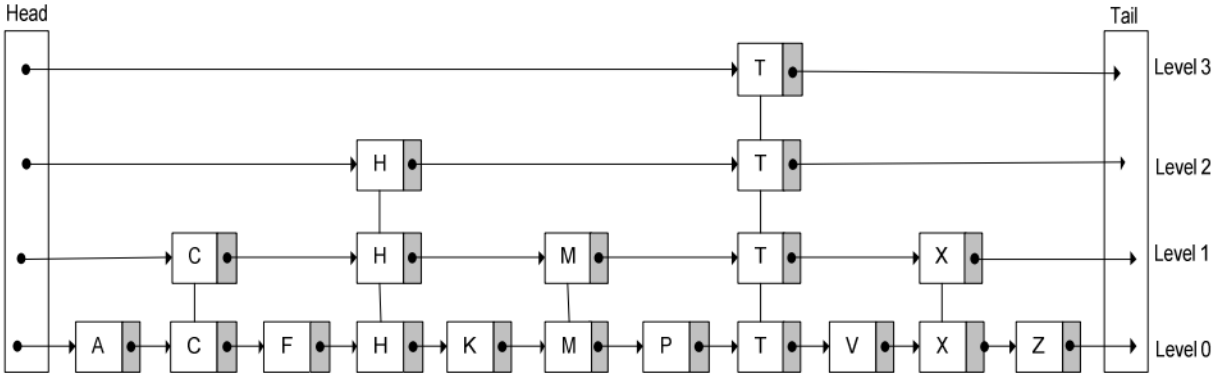

Figure 3. Standard Skip list

3. Improved Circular Skip List(ICSL)

As described above, by using the circular linked list and skip list data structures, a new data structure called Improved Circular skip list (ICSL) was developed. Operations are performed only on a single ordered list in circular linked list (Figure 2). However, in our method, nodes are searched, inserted and deleted on circular linked lists (Figure 4) which are linked to each other in levels that are indexes of each other.

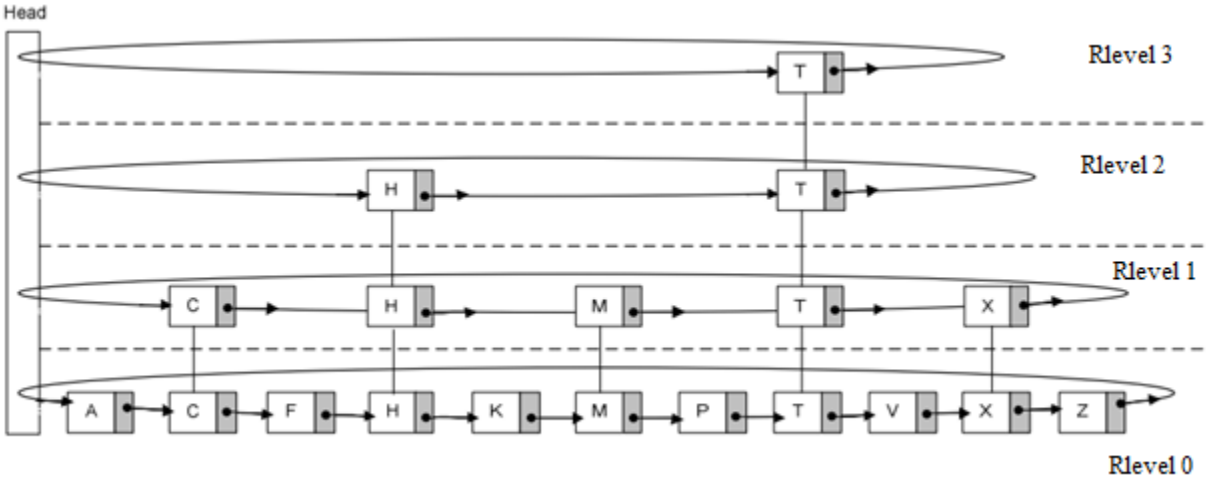

Figure 4. Improved circular skip list

In this new conical data structure, the relationships are defined as Circular linked list 0 = RLevel 0, Circular linked list 1 = RLevel1, ... , Circular linked list (l)=RLevel (l) (Figure 4). In ICSL data structure each ring is a sub-sequence of the previous one, RLevel 0 ⊇ RLevel 1 ⊇... ⊇ RLevel l. RLevel 0 at the bottom level in improved circular skip list data structure and encompasses all elements. Each ring level from bottom to the top is lined as an index

of previous ring called Rlevel. In addition, ICSL data structure is similar to skip list data structure. In contrast to skip list data structure, head and tail is not required together and only head is enough in ISCL data structure. Our new data structure is created by removing the tail from skip list data structure. In this way the last element (tail) of circular linked list is linked in a way that it points back to the first element (head) (Figure 4), and this process is performed for entire levels.

When RLevel in ICSL data structure are created (RLevel 0, RLevel 1,.., RLevel l), levels are created randomly. Let us say that the number of ordered nodes in our ICSL data structure is N. RLevel 0 consists of these entire N ordered nodes (Figure4 Rlevel 0). Because N ordered elements are included, search at RLevel 0 level is performed in O(N) time complexity. Rlevel 1 is created if every other element of the list at RLevel 0 also has an extra link to the element two ahead of it (Figure 4 – RLevel 1). Since the maximum number of elements at RLevel 1 level equals $\lceil \frac{N}{2} \rceil + 1$, search is performed within O(N) time complexity. RLevel 2 is created if every forth element of the list at RLevel 0 also has an extra link to the element four ahead of it (Figure 4 RLevel 2). Since the maximum number of elements at Rlevel 2 level equals $\lceil \frac{N}{4} \rceil + 2$. , search is performed within O(N) time complexity. When each 2^ith node (i=0,..,MaxLevel(15 or 31)) is linked to the following 2^ith node via a pointer in this way, since the maximum number of elements at RLevel i level equals $\left\lceil \frac{N}{2^i} \right\rceil + i$, time complexity to reach a node in a search equals O(log N) at maximum. Similar to skip list data structure searching, insertion and deletion [4,6] can also be performed in our ICSL data structure.

3.1. Searching in ICSL

In our ICSL data structure, as described in the previous section, each 2^ith (i=0,..,MaxLevel(15 or 31)) node is linked to the following $2i^{th}$ node via a pointer. Thus, a conical structure is created, which allows reaching the required node in a time complexity of O(log N) at maximum. Search is initiated in the Rlevel at the top level and continues towards Rlevel at lower levels. Steps to be followed for search in our new data structure are as follows:

Algorithm 1(searching in ICSL)

```
Searchnode(Rlevel,key)
Temp←rlevel.head
Level←Rlevel
If(temp.next[0]=Rlevel.head) or level<0)
Return false
For I ←level downto 0 do
While(temp.next[i]≠Rlevel.head
And temp.next[i]→value < key)
Temp←temp.next[i]
Temp←temp.next[0]
If(temp≠Rlevel.head and temp.value=key)
Return true;
Return false;
```

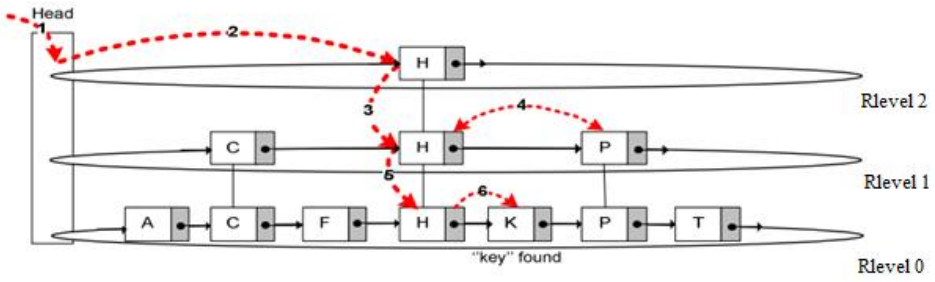

Figure 5: Searching in ICSL, find 'K' node in a ICSL consisting of {A, C, F, H, K, P, T} elements

Because it is similar to skip list data structure, search in ICSL data structure can be performed through modifying the algorithm used in skip list.

3.2. Insertion in ICSL

Algorithm 3: Insertion in ICSL

```
InsertNode(Rlevel,key)
Temp←Rlevel.head
Level←Rlevel.level
Update[maxlevel +1]
For I ←level down to 0 do
While (temp.next[i]≠Rlevel.head
And temp.next[i].value < key
Temp←temp.next[i]
Update[i]←temp;
End for
Temp←temp.next[0];
If(temp_Rlevel.head or temp.value≠key)
(generation new level by random_level() algorithm)
Newlevel ←random_level();
If (newlevel>level)
For I level+1 to newlevel do
Update[i]←temp;
Level←newlevel;
Endif
Temp←make_newnode(newlevel,value);
For i←0 to newlevel do
Temp.next[i]←update.next[i];
Update.next[i]←temp;
End for
End if
```

It is required to find the position in order to insert a new node, which requires searching. It is possible to reach a node in a time complexity of O(log N) at maximum, which is the time complexity for node insertion. While constructing skip ring data structure from nodes, nodes are placed at random levels. The random_level() algorithm (Algorithm 1) creates a random level between 0,..,MaxLevel(15 or 31) to form up levels to insert nodes.

Algorithm 2: Generating random level

```
Random_level()
Rlevel←0;
Frand←rand()
{frand value in 0..1}
While (frand<p) and (level<maxRlevel)
{p=1/2 or 1/4}
Rlevel←Rlevel+1;
return level;
```

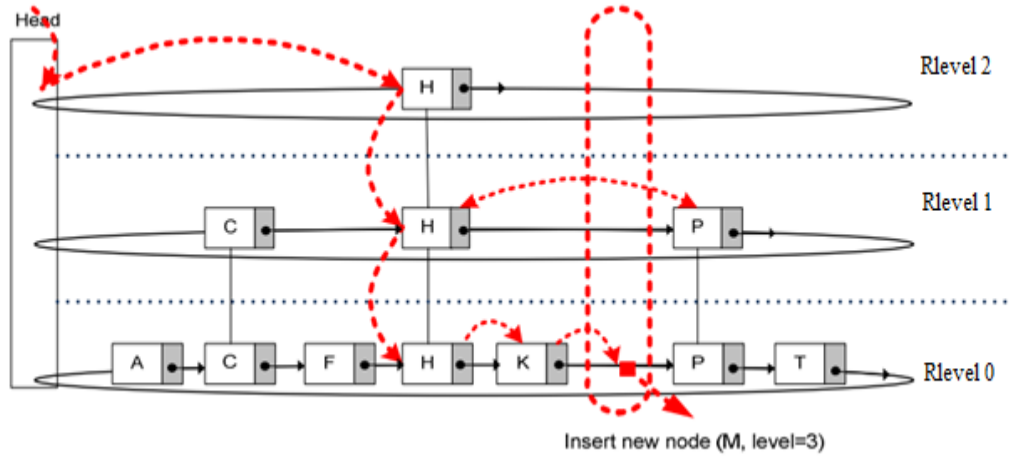

Figure 6(a): The steps to insert 'M' node (random level=3) in a ICSL data structure

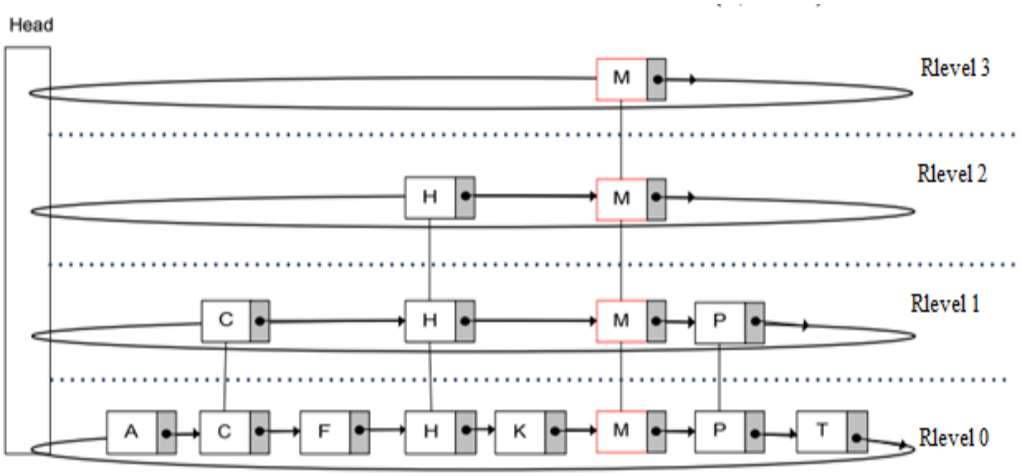

Figure 6(b). Insert new node and update ICSL

3.3 Deletion in ICSL

It is required to search and find a node in order to delete a node. It is possible to reach a node in a time complexity of O(log N) at maximum, which is the time complexity for node deletion. Deletion of a node in a ICSL data structure is similar to that of a skip list data structure [4,6]. Similar to search algorithm, the first thing that must be controlled during removal operation is that not a single element can be available in ICSL data structure.

Algorithm 4 (Delete node in Improved circular skip list

```
DeleteNode(RLevel,key)
Temp←Rlevel.head
Level←Rlevel
Update(MaxLevel+1);
If(temp.next[0]=Rlevel.head)or
    (level<0)
     Returns false
For i←Rlevel downto 0 do
     While (temp→next[i]≠Rlevel.head
          And temp.next[i]→value<key
      Temp←temp.next[i];
    Update[i]←temp;
End for
Temp←temp.next[0];
If (temp.value=value)
    For i← 0 to Rlevel do
      If (update.next[i]≠temp)
        Break;
        Update.next[i]=temp.next[i]
    End for
Free(temp);
While(Rlevel>0 and
  temp.next[Rlevel]=rlevel.head)
    Rlevel←Rlevel-1;
End if
```

3.4 ICSL Priority Search

The innovative search algorithm which was called priority search used in the ICSL data structure. It was benefited from the pyramidal layered-structure of the skip list data structure. The standard searching algorithm (algorithm 1) in the skip list data structure starts at top-level to the lowest level until it finds the searching data or it ends up in the lowest level. In this we developed new searching algorithm for ICSL (Algorithm 5 was based on the hit search number for each searched data. If a datum has greater hit search number, then it was upgraded in the skip list data structure to upper level. That is, the mostly searched data were located in the upper levels of the skip list data structure and rarely searched data were located in the lower levels of the skip list data structure. The time complexity of searching in the skip list data structure (Algorithm 1) is O(logN), but the time complexity of searching algorithm in priority search (Algorithm 5) approximates to Θ(1). In another word, the mostly searched data were located in the top-level of the skip list data structure, thus, the searching for these data has time complexity as Θ(1). The rarely searched data were located in the lowest level and their searching time complexities approximate to O(logN). The time complexity of searching by using priority search algorithm changes between Θ(1)- O(logN). Table I will be obtained by using the priority search algorithm. It was performed by using frequencies (hit search numbers). That is, the searched data is upgraded once for each search process. Therefore, the mostly searched data were located at the top of skip list data structure (pyramidal structure) and rarely searched data were located at the bottom of skip list data structure. The priority search algorithm was

Nodes	**A**	**C**	**F**	
Frequency	0	1	0	
level	0	1	0	
Nodes	**H**	**K**	**P**	**T**
frequency	2	0	1	0
Level	2	0	1	0

Tabel 1. Frequency wise level distribution of nodes in ICSL

Algorithm 5: Priority search with ICSL

```
PrioritySearch(Rlevel,Search_Key)
Head=Rlevel_head
Level←Rlevel
Update[maxlevel +1]
While(Rlevel≠0)
If(head→next[Rlevel]→value=Search_key)
For i←Rlevel downto 0 do
  While(head.next[i]≠null and head.next[i]=value < search_key)
  Head←head.next[i]
  Update[i]←head
End for
Head←head.next[0]
Intlevel=Rlevel+1;
if(level>Rlevel)
update[level]=Rlevel.head
Rlevel=level
End if
Head→next[level]= update[level→next[level]
Update[level]→next[level]=head
Return true
End if
If (head→next[level]→value < search_key)
Head=head-->next[level]
If(head→next[level→valur > search_key)
Level=level-1
End while
Return false;
```

used in the skip list data structure due to its pyramidal structure. Additionally, the standard search algorithm (Algorithm 1) for the skip list of size N has time complexity as O(logN). Data were sorted in ascending order in the skip list data structure when skip list data structure were constructed. The most important property of skip list data structure is its pyramidal structure and the internet. If data were located in a large skip list data structure for search engine, it would be more advantageous. This data structure is also advantageous for dictionary operations, since the most hit data will be on ordered data in it. The searching process started at the first element in the list and carried on till the end of list, when data were unordered. So, the searching algorithm is a linear algorithm in term of the number of data in the list. The time complexity of linear search is O(N). If data were unordered, initially they must be ordered by using any sorting algorithm.

Significance of using Priority search algorithm in ICSL locates the most searched data to the top of the pyramid-shaped skip list data structure. For these reason, enabling time complexity $\Theta(1)$ of frequent searched data were important. The priority search algorithm may be used in the search engine like Google, Yandex, etc. The greater frequency (search hit number) the upper level for searched data; the smaller frequency the lower level for searched data. The mostly searched data were located in the top level of skip list data structure, so, searching this data will take less time. The rarely searched data were located in the lowest level of the skip list data structure, so, its searching time will take longer. If searching process was grouped with respect to frequencies of data, the searching would be easier. There many data (may be billion data, etc.) in

the top level of skip list data structure and its searching will take shorter time; the least hit data will be on the lowest level of the skip list data structure and its searching time will take longer time.

4. EXPERIMENTAL RESULTS: PRIORITY SEARCH AND OTHERS

The proposed algorithm was implemented by using java and tested successfully. In order to compare ICSL Priority Search (ICSL PS), Circular Skip list search (CLS), and Standard Skip List Search (SSL), random and sorted data were used. The searching times of ICSL PS, CLS and SL for sizes from 1000 to 50000 of data in list were illustrated in the Table II and Table III. Moreover, each algorithm was applied to same size list 100 times and all times for all executions was added up and then their average was computed. This means that the effect of data permutation will be minimized and the comparison will be more equitable. If there is one search for algorithm, the comparisons may be non-equitable. For example, searched data for PS may be on the top level of skip list data structure, and then its time will be Θ(1). If the searched data for CLS is not found in the circular skip list, then its time will be longer. This case may be available for each search algorithm. Due to this case, there were 100 executions for equitable comparisons of search algorithms. All results were obtained on the same computer and the results in Table II and Table III demonstrated that when size of list is small, CSL shows normal performance; when the size of array increases, the performance of ICSL PS increases and PS is better than CLS and SSL. The results were illustrated in Figure. 7

Table II Searching Time for ICSL PS, CSL and SSL for sorted list data (ms =milliseconds)

No. of Nodes	**1000**	**5000**	**100000**	**30000**	**50000**
SSL	0.0033	0.0105	0.0170	0.0400	0.0400
CSL	0.00015	0.00017	0.00021	0.00024	0.00026
ICSL	0.00009	0.00011	0.00012	0.00016	0.00018

Table II, Table III and Fig. 7, Fig. 8 depict that ICSL PS is better than SSL and CSL with respect to searching time. The time complexities for searching ICSL PS, and CSL on sorted lists are O(logN). The time complexities for searching SSL on sorted list is O(log N). While computing time complexity for any algorithm, the dominant (term with the greatest degree) term is regarded as time complexity. The asymptotic behaviors of ICSL PS and CSL are similar; however, the constant coefficients are different and this case makes ICSL PS be the best algorithm. It is noticeable in Table II and Table III; ICSL PS algorithm has better performance than C LS and SSL. Moreover, ICSL PS algorithm is better than CLS algorithm as seen in Fig. 8. Searched data in CLS PS algorithm were located to the top of Skip List, hence time complexity will be Θ(1) for these data.

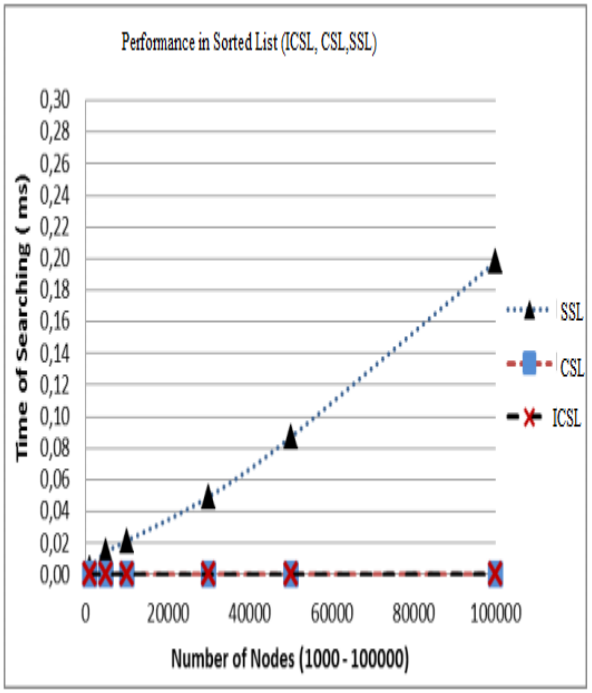

Figure 7. Performance comparison for SSL, CSL, ICSL PS (If the searched data are in middle of List)

TABLE III. SEARCHING TIMES FOR SSL, CSL AND ICSL PS FOR SORTED LIST (IF THE SEARCHED DATA ARE NEAR TO THE END OF List) (MS=MILLISECOND)

No of Nodes	1000	10000	30000	50000
SSL	0.0048	0.0173	0.0850	0.2888
CSL	0.00014	0.000016	0.00020	0.00024
ICSL	0.00008	0.00010	0.00015	0.00017

The results in Table II were obtained when the searched data were located near to the beginning of List. Whereas, Table III shows the situation where the searched data were located near to the end of the List. Comparing the results of algorithm in both tables, it was seen that the search time increases if the data were located SS Lat the end of List. However, the results were the same for CSL and ICSL PS algorithms no matter where the searched data was located.

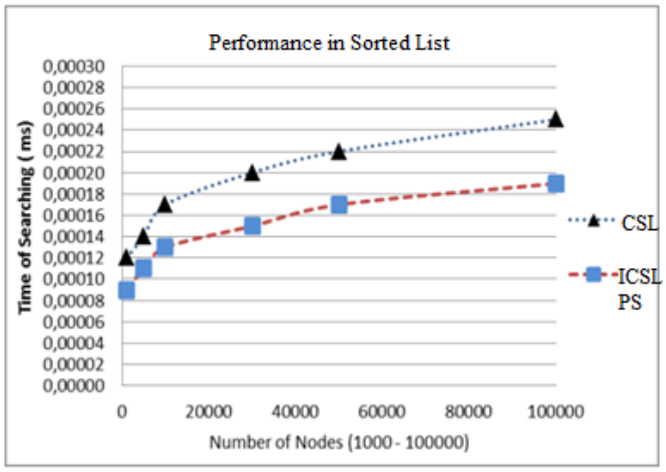

Figure 8. Performance comparison for ICSL PS and CSL (Sorted List)

5. CONCLUSION:

Skip list data structure was created with the help of circular linked list data structures. Due to the layered structure, ICSL data structure presented in this study reduces the time complexity of search, insertion and deletion processes in linked list data structure to O(logN), which was O(N). The improved priority searching was better than searching in standard skip list considering the applications. The time complexity of priority search algorithm was between Θ(1)-O(logN); the most searched data has time complexity as Θ(1), the least searched data has time complexity as O(logN). To summary priority search algorithm could be used in searching processes more efficiently.

It enables saving remarkable time when larger sets of data were handled.

6. References

[1] Aksu, M.; Karcı, A.; Yılmaz, ¸ S.: Level optimization in Skip List data structure. In: Proceedings of the 1st International Symposium on Innovative Technologies in Engineering and Science (ISITIES2013). pp. 389-396 (2013)
[2] Aksu, M.; Karcı, A.; Yılmaz, ¸ S.: Effects of P threshold values in creation of random level and to the performance of skip list data structure. Bitlis Eren Univ. J. Sci. **2**(2), 148–153 (2013)
[3] Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C.: Introduction to Algorithms. MIT Press, London (2009)
[4] McMillan, M.: Data Structures and Algorithms Using C#. Cambridge University Press, New York (2007)
[5] Shaffer, C.A.: Data Structures & Algorithm Analysis in C++. Dover Publications, Mineola (2011)
[6] Colvin, R.; Groves, L.; Luchangco, V.; Moir, M.: Formal verification of a lazy concurrent list-based set. In: Proceedings of the Computer Aided Verification, Lecture Notes in Computer Science. 4144, 475–488 (2006)
[7] Herlihy, M.; Lev, Y.; Luchangco, V.; Shavit, N.: A simple optimistic skiplist algorithm. In: Proceedings of the Structural Information and Communication Complexity. Lecture Notes in Computer Science. 4474, pp. 124–138 (2007)
[8] Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Commun. ACM **33**(6), 668–676 (1990)
[9] Kirschenhofer, P.; Prodinger, H.: The path length of random skip lists. Acta Inf. **31**(8), 775–792 (1994)
10. Papadakis, T.: Skip lists and probabilistic analysis of algorithms. PhD Thesis. University of Waterloo. Tech. Report CS-93-28, (1993)
[11] Poblete, P.V.; Munro, J.I.; Papadakis, T.: The binomial transform and the analysis of skip lists. Theor.Comput. Sci. **352**, 136–158 (2006)
[12] Munro, J. I.; Papadakis, T.; Poblete, P.V.: Deterministic skip lists. In: Proceedings of the SODA ’92 Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms. pp. 367–375 (1992)
[13] Pugh, W.: A Skip List Cookbook. Dept. of Computer Science, University of Maryland. College Park. Technical report. CS–TR– 2286.1. (1990)
[14] Pugh, W.: Concurrent Maintenance of Skip Lists. Dept. of Computer Science. University of Maryland. College Park. Technical report. TR– 2222.1. (1989)
[15] Lotan, I.; Shavit, N.: SkipList-Based Concurrent Priority Queues. In: Proceedings of International Parallel and Distributed Processing Symposium. Mexico, pp. 263–268 (2000)