Lab Manual

CL2001 – Data Structures

Fall-2023

National University of Computer and Emerging Sciences-FAST
Karachi Campus

Data Structures Lab#4
Course: Data Structures (CL2001) **Semester:** Fall 2023
Instructor: Shafique Rehman **T.A:** N/A

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise your hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session**.**

Contents:

- Shell Sort
- Singly Linked List
- Doubly Linked List
- Circularly Linked List

Shell Sort:

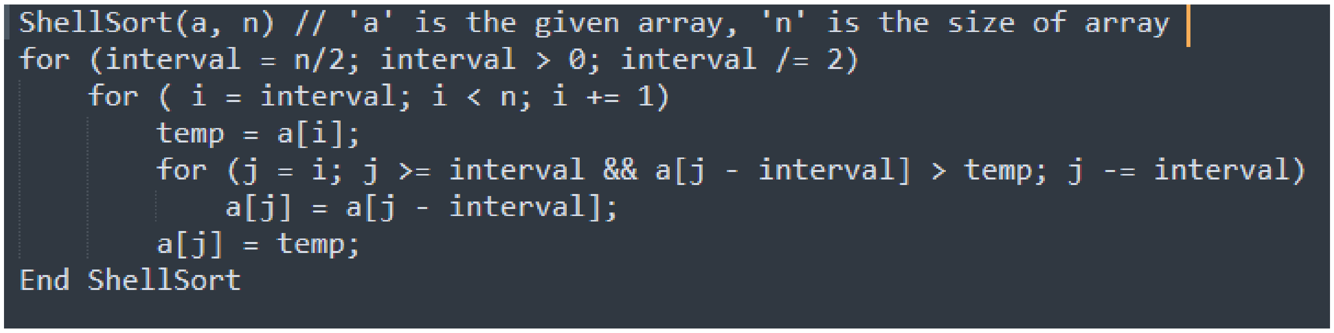
Shell sort is the generalization of insertion sort, which overcomes the drawbacks of insertion sort by comparing elements separated by a gap of several positions.

It is a sorting algorithm that is an extended version of insertion sort. Shell sort has improved the average time complexity of insertion sort. As similar to insertion sort, it is a comparison-based and in-place sorting algorithm. Shell sort is efficient for medium-sized data sets.

In insertion sort, at a time, elements can be moved ahead by one position only. To move an element to a far-away position, many movements are required that increase the algorithm's execution time. But shell sort overcomes this drawback of insertion sort. It allows the movement and swapping of far-away elements as well.

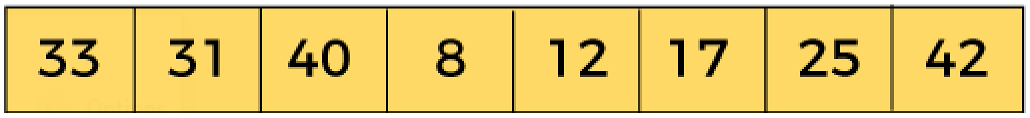
This algorithm first sorts the elements that are far away from each other, then it subsequently reduces the gap between them. This gap is called as interval.

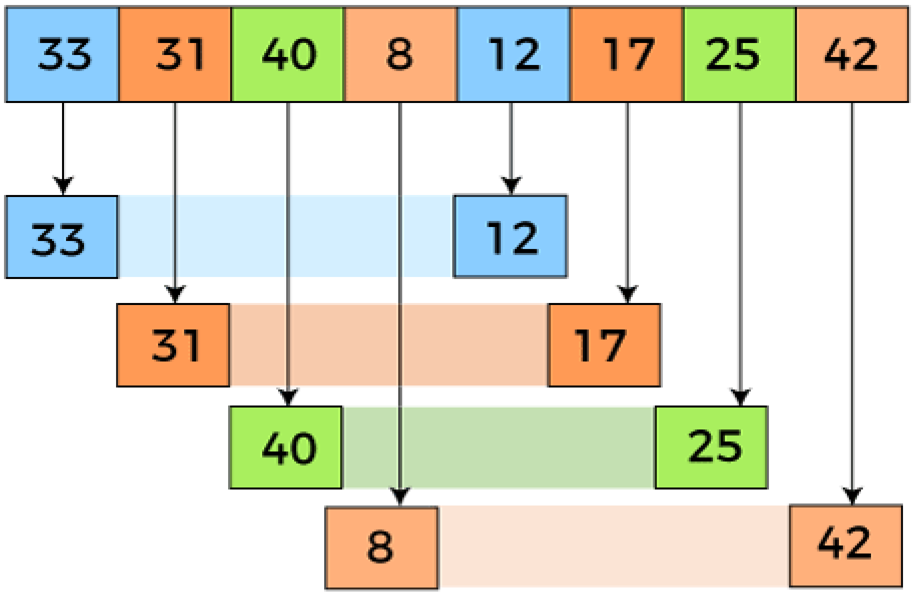
Algorithm:

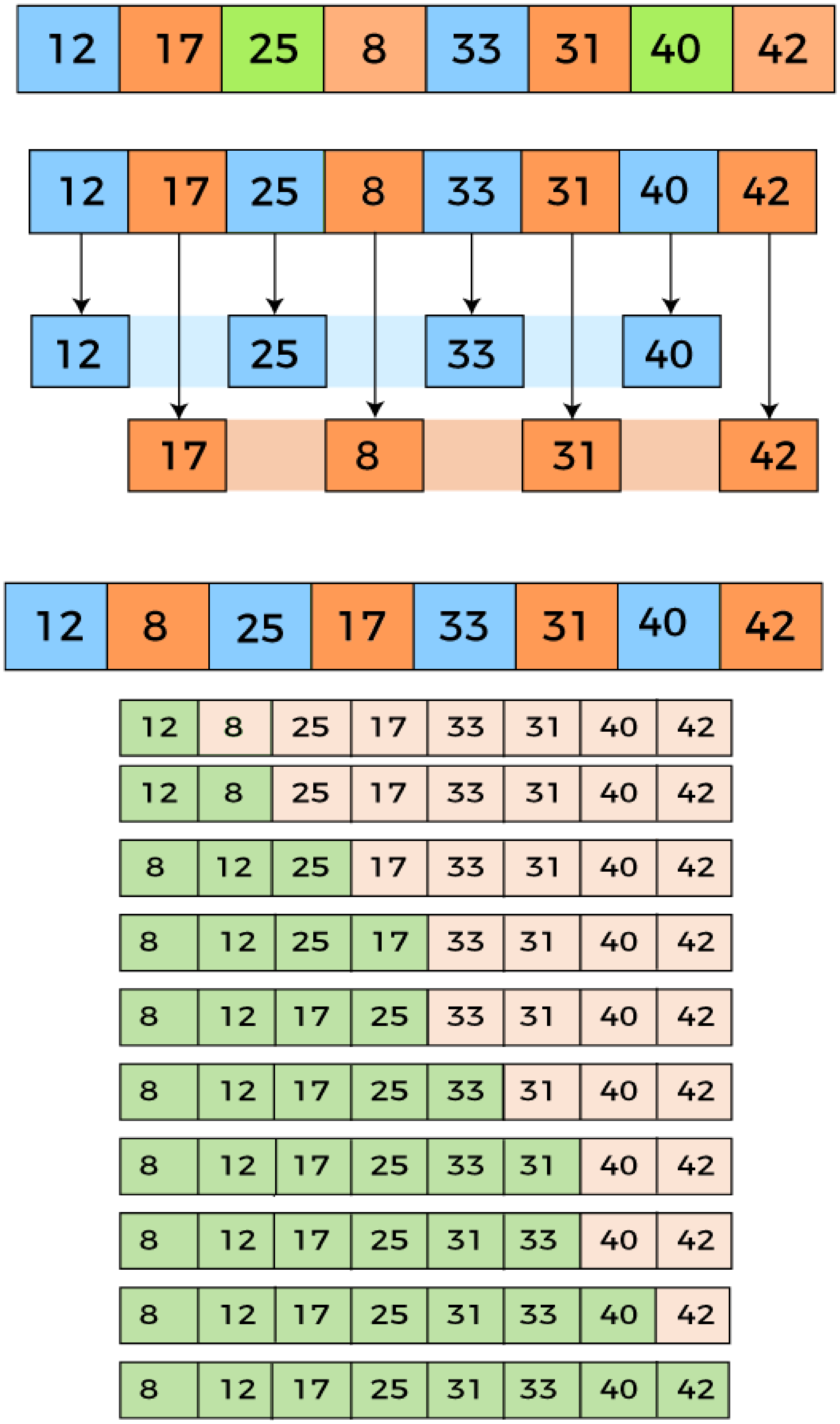


```
ShellSort(a, n) // 'a' is the given array, 'n' is the size of array
for (interval = n/2; interval > 0; interval /= 2)
    for ( i = interval; i < n; i += 1)
        temp = a[i];
        for (j = i; j >= interval && a[j - interval] > temp; j -= interval)
            a[j] = a[j - interval];
        a[j] = temp;
End ShellSort
```

Working of Shell Sort:






12 17 25 8 33 31 40 42
12 17 25 8 33 31 40 42
12 25 33 40
17 8 31 42
12 8 25 17 33 31 40 42
12 8 25 17 33 31 40 42
12 8 25 17 33 31 40 42
8 12 25 17 33 31 40 42
8 12 25 17 33 31 40 42
8 12 17 25 33 31 40 42
8 12 17 25 33 31 40 42
8 12 17 25 33 31 40 42
8 12 17 25 31 33 40 42
8 12 17 25 31 33 40 42
8 12 17 25 31 33 40 42

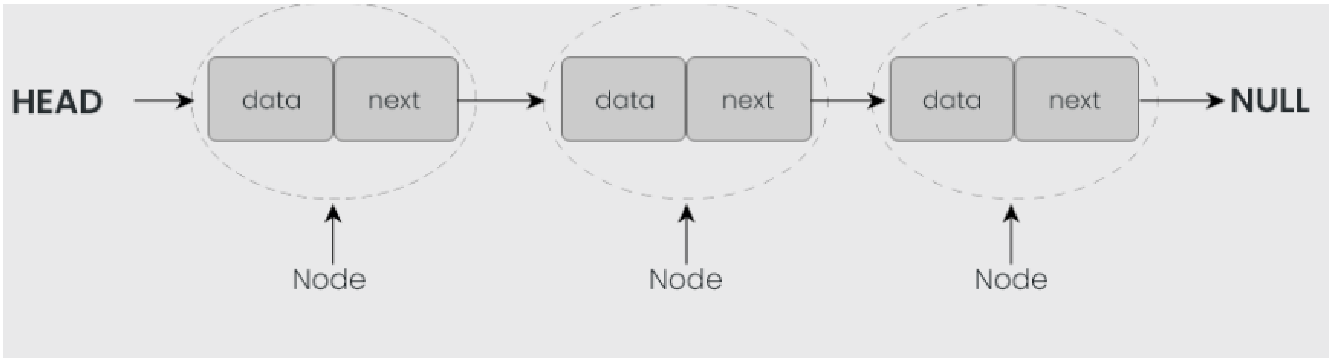
Linked List:

We saw that arrays had certain disadvantages as data storage structures. In an unordered array, searching is slow, whereas, in an ordered array, insertion is slow. In both kinds of arrays, deletion is slow. Also, the size of an array can't be changed after it's created.
Linked lists are probably the second most commonly used general purpose storage structures after arrays.

Linked list has the following properties

- Successive elements are connected by pointers/references
- The last element points to NULL
- Can grow or shrink in size during execution of a program
- Can be made just as long as required (until systems memory exhausts)

The linked list is a versatile mechanism suitable for use in many kinds of general-purpose databases. It can also replace an array as the basis for other storage structures such as stacksand queues. In fact, you can use a linked list in many cases in which you use an array unless you need frequent random access to individual items using an index. A simple linked list is shown below:



The linked list shown above is called as **Singly** or **Single-Ended Linked List** because we have the reference of only one end called the **first** (which refers/points to the front end of the list) and can move only **forward** because each node has the reference of next node only.

Practice Task 2

Add method (in Practice Task 1) for Deleting the first item and Deleting All item in the list.

Practice Task 3

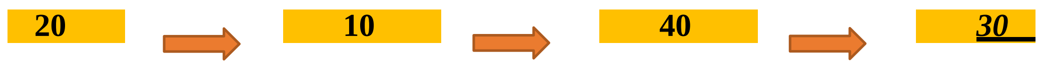
a) Add two more methods (in Practice Task 1) one for **finding** any given item another for **deleting** any specific item in LinkedList. For example, find 30 from the following list and then delete it from the list.

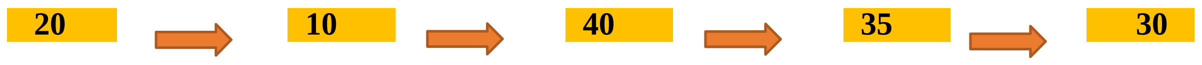
$$\begin{tikzcd}\mathbf{10} \arrow[r] & \mathbf{20} \arrow[r] & \underline{\boldsymbol{30}} \arrow[r] & \mathbf{40}\end{tikzcd}$$

- Updated list after deleting 30

$$\begin{tikzcd}\mathbf{10} \arrow[r] & \mathbf{20} \arrow[r] & \mathbf{40}\end{tikzcd}$$

b) Create another method to insert 35 before 30 as shown in following two figures.





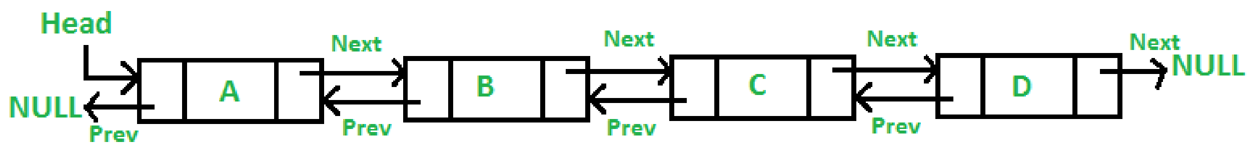
Double-Ended Linked List

A double-ended list is similar to an ordinary linked list, but it has one additional feature: a reference to the last link as well as to the first link.

The reference to the last link permits you to insert a new link directly at the end of the list as well as at the beginning. Of course, you can insert a new link at the end of an ordinary single- ended list by iterating through the entire list until you reach the end, but this approach is inefficient. Access to the end of the list as well as the beginning makes the double-ended listsuitable for certain situations that a single-ended list can’t handle efficiently. One such situation is implementing a queue.

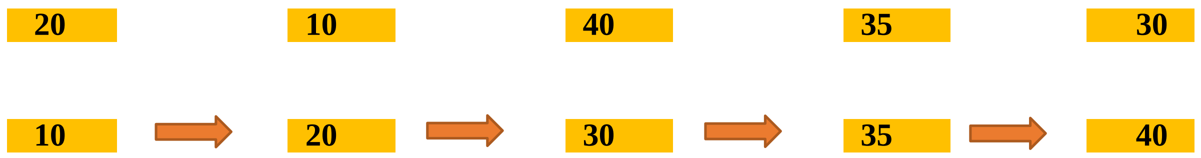
Doubly Linked List

There is a third type of list called the doubly linked list (not to be confused with the double-ended list). What’s the advantage of a doubly linked list? A potential problem with ordinary linked lists is that it’s difficult to traverse backward along the list. The doubly linked list provides this capability. It allows you to traverse backward as well as forward through the list. The secret is that each link has two references to other links instead of one. The first is to the next link, as in ordinary lists. The second is to the previous link. This type of list is shown in the following figure:



Practice Task 1: Sorted Linked List Example

Practice Task 2: Use Sorted Linked List as a sorting mechanism to sort the followingunordered array into an ordered array, as shown below.



Practice Task 3: Doubly Linked List.

Practice Task 4: Write Java code to include following methods in practice task 3:

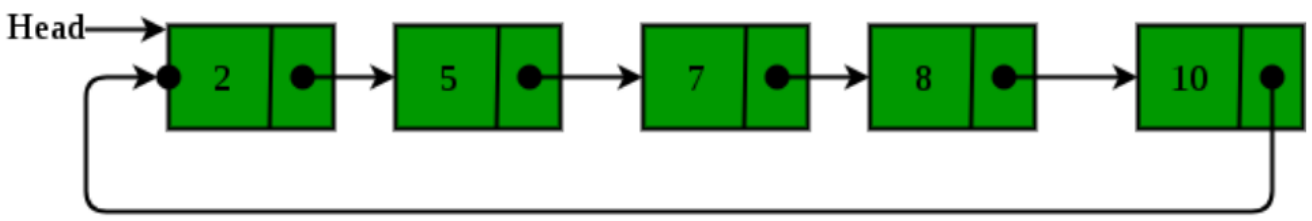
a) insertLast()
b) deleteFirst()
c) deleteLast()
d) deleteKey(long key)
e) displayBackward()

Circular Linked List

Circular Linked List is a variation of the Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

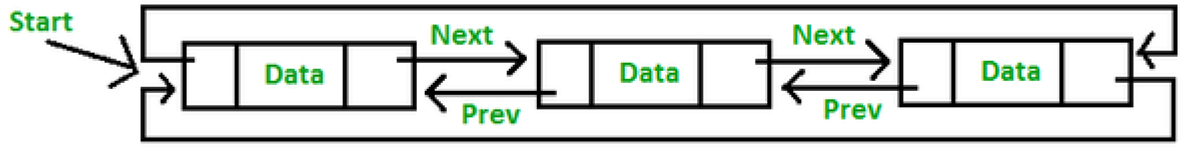
Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node



Doubly Linked List as Circular

In the doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, the following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as the doubly linked list.

- The first link's previous points to the last of the list in case of a doubly linked list.

Basic Operations

Following are the important operations supported by a circular list.

- **insert** − Inserts an element at the start of the list.
- **delete** − Deletes an element from the start of the list.
- **display** − Displays the list.