DATA STRUCTURES ASSIGNMENT-II

19BQ1A05L1
SK. Vaseem Naazleen
II CSE - D

Q1: Explain any external sorting method with suitable example. * shell sort is in-place sorting algorithm.

Ans: * Shell sort is mainly based on insertion sort.

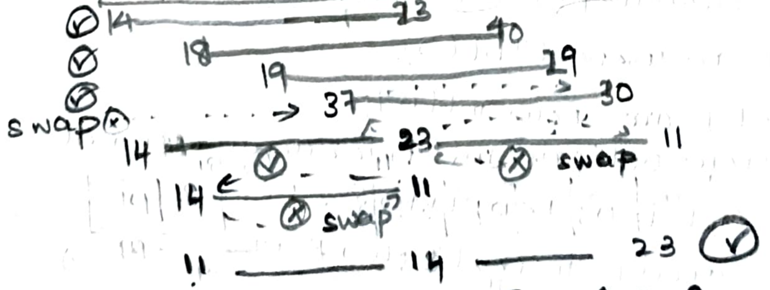
* In insertion sort, we move elements only 1 position ahead. When an element has to be moved far ahead, many movements are involved.
* The idea of shell sort is to allow exchange far items.
* The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements compared.
* By starting far apart elements. It can move some out of place elements into positions faster than a simple nearest neighbour exchange
* Run time of shell sort is heavily dependent on the gap.
* Time complexities :

Best case: $O(n\log n)$ Avg case: $O(n(\log(n))^2)$ worst case: $O(n^2)$

* Example:

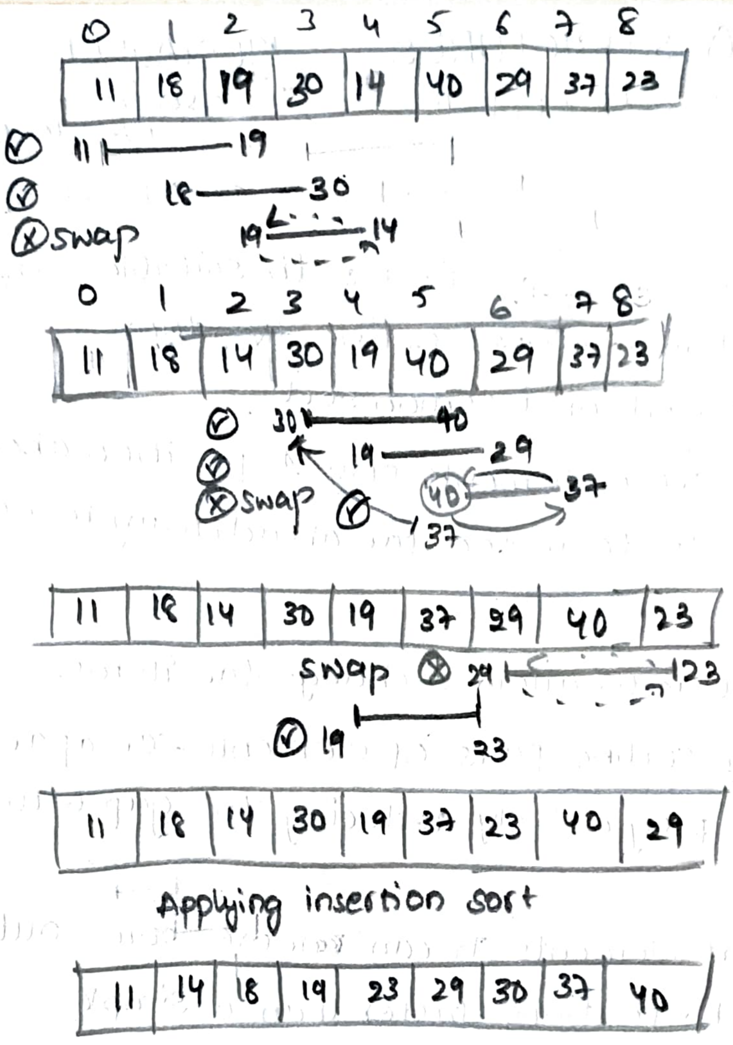
0	1	2	3	4	5	6	7	8
14	18	19	37	23	40	29	30	11

$gap = \frac{n}{2} = \frac{9}{2} = 4$



0	1	2	3	4	5	6	7	8
11	18	19	30	14	40	29	37	23

$gap = \frac{4}{2} = 2$



gap=2

when swap check prev elems with same gap

gap=1 (same as insertion sort)

Applying insertion sort

Array is sorted.

Program:

```
import java.io.*;
import java.util.Scanner;

public class ShellSort {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                System.out.println("Enter size of array:  ");
                int n=in.nextInt();
                int[] arr= new int[n];
                System.out.println("Enter array elements:  ");
                for (int i=0; i<n; i++) {
                        arr[i]= in.nextInt();
                }
                shellsort(arr);
        System.out.println("Sorted array:  ");
        for(int i=0; i<n; i++){
            System.out.print(arr[i]+"    ");
        }
        System.out.println();
    }
    public static void shellSort(int[] arr){
            int temp, n=arr.length();
            for(int gap=n/2; gap>0; gap/=2){
                for(int i=gap; i<n; i++){
                    for(int j=i-gap; j>=0; j-=gap){
                        if(arr[j+gap] > arr[j]){
                            break;
                        } else {
                            temp= arr[j+gap];
                            arr[j+gap] = arr[j];
                            arr[j] = temp;
                        }
                    }
                }
            }
    }
}
```

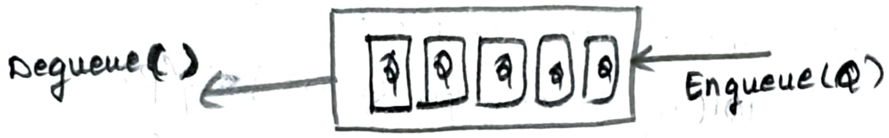
Output

```
Enter size of the array:
 9
Enter array elements:
14  18  19  37  23  40  29  30  11
Sorted array:
11  14  18  19  23  29  30  37  40
```

Q2. Give the structure Queue ADT. Explain the operations on it.

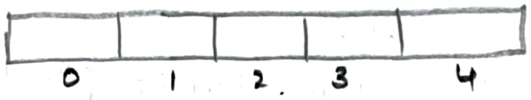
Queues: Like Stacks, Queues are Lists. With a queue, However, Insertion is done at one end called rear end, where as deletion is performed at the other end called front end. Sometimes it is called FIFO (First in first out) List.

The basic operations on a queue are Enqueue, which inserts an element at the end of the list (called the rear) and Dequeue, which deletes the elements at the start of the list.



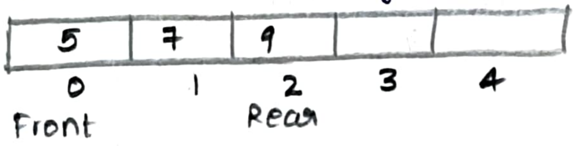
Array Implementation of queue: Consider a queue of maximum 5 elements.

After create

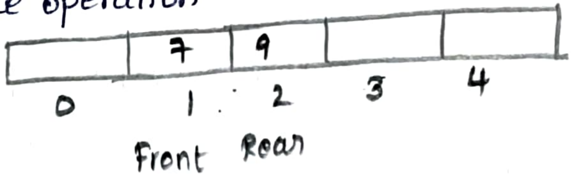


Front = 0, Rear = 0

After adding 5, 7, and 9 in same sequence the queue will be



After an Dequeue operation

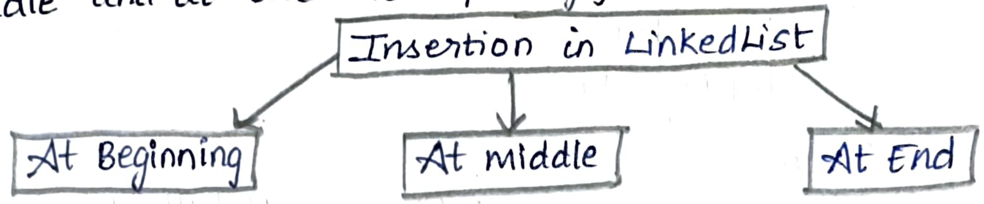


Program:

```
import java.util.Scanner;
class Queue1 {
        int max, f, r;
        int q[];
        Queue1()
        {
          f = r = -1;
          max = 5.
          q = new int[max];
        }
```

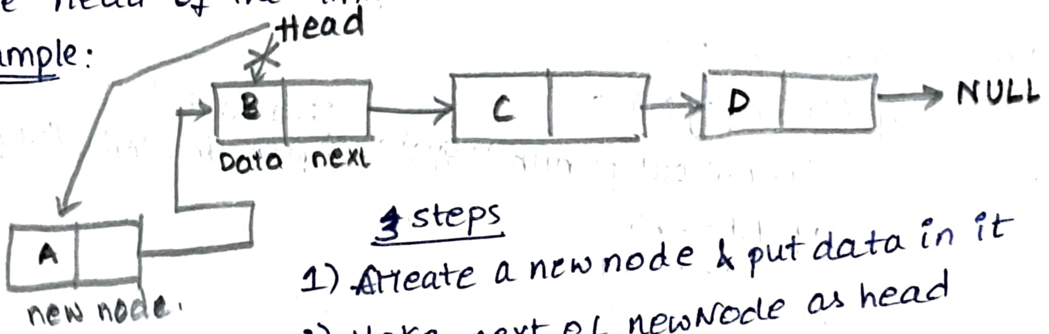
```
    void   enQueue(int e){
           if (!isFull()){
                 q[++r]=e;
                 if (f==-1)
                      f++
           } else {
                System.out.println("queue is full ");
           }
    }

    void   deQueue () {
           if (!isEmpty()) {
                System.out.println(" the element deleted : "+q[f]);
                if (f==r)
                     f=r=-1;
                else
                     f++;
           }
           else {
                System.out.print ln("Queue is empty");
           }
  }
  boolean isEmpty(){
          return  f== -1;
  }

 boolean  is Full() {
          return  r==max-1;
 }

 void  display () {
        if(!isEmpty()){
              for(int i=f ; i<=r ;i++){
                   System.out.print(q[i]+ "   ");
              }
              System.out.println();
        }
        else{
          System.out.printnl("Queue is Empty").
 }      }
}
```

Q3: Explain an algorithm the insert new node at beginning, at middle and at the end of singly Linked List.



1) At Beginning: The new node is added at the front and it becomes the head of the linked list.

Example:



3 steps

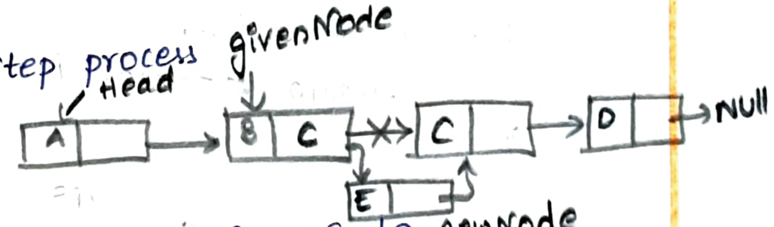
1) Create a new node & put data in it
2) Make next of newNode as head
3) Make newNode as head.

Code:

```
public void insertAtBeginning(int data) {
        Node newNode = new Node(data); //1
        newNode.next = head; //2
        head = newNode //3
}
```

2) At middle / given position: 4 step process
or after node

1) Check if given node is null



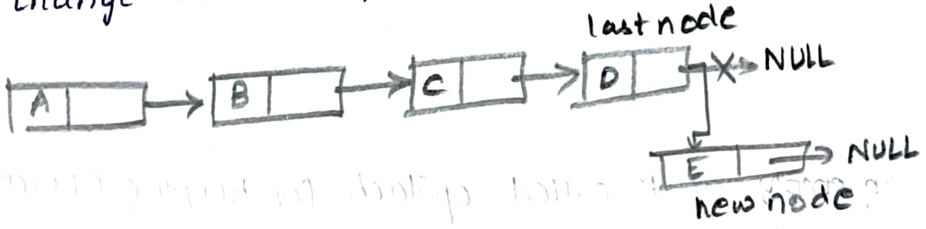
2) Create new node
3) Make next of newNode as next of ~~prev~~ givenNode
4) Make givennode's next as newNode.

Code:

```
public void insertAfterNode(Node prevNode, int data) {
                if (prevNode == null) return; //1
        Node newNode = new Node(data); //2
                newNode.next = prevNode.next; //3
                prevNode.next = newNode; //4
}
```

3) <u>At end</u>: 5 step process

1) Create newNode
2) It is going to be last node. So, point its next to null
3) Check if head is empty, if it is then make newnode as head
4) else traverse till end of list (go to last node)
5) Change the next of last node to point to newNode



<u>Code:</u>

```
public ~~static~~ void insertAtEnd (int data){
        Node newNode = new Node(data); //1
        newNode.next = Null; //2
        if (head == null) { //3
            head = newNode;
            return;
        }

        Node last = head;
        Whlie (last.next != null) {     //4
            last = last.next;
        }

        last.next = newNode; //5
}
```

Q1: Illustrate the algorithm to conversion of infix expression to postfix with an example

Example: $(A + (B * C)) \longrightarrow A\ B\ C * +$

Infix expression   Postfix expression

$(A + B) * C \longrightarrow A\ B + C *$

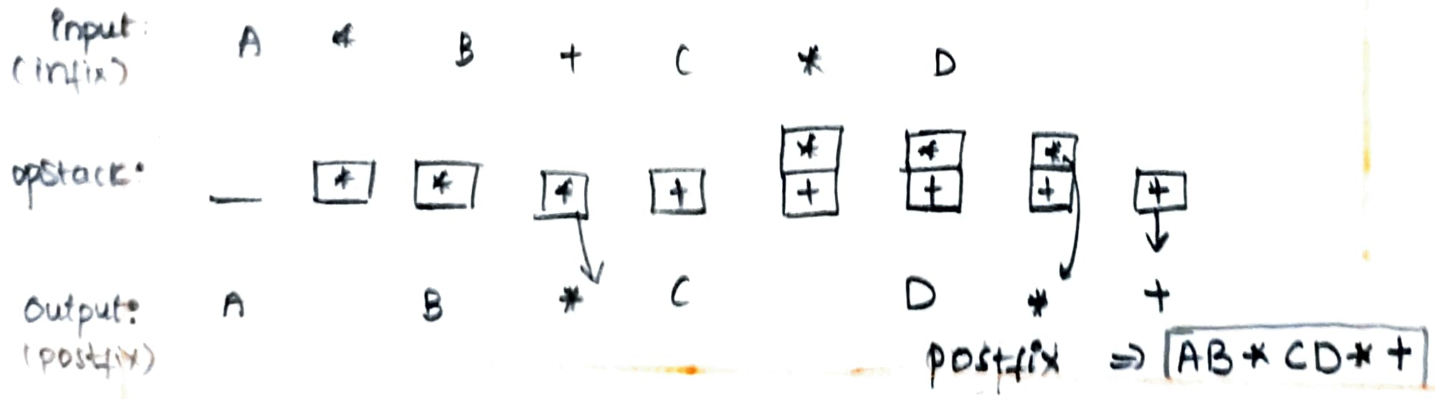
Algorithm:

1. create an empty stack called opStack for keeping operators. create an empty string for output.

2. ~~convert the input infix string to a list~~ Scan input infix String from left to right
   → If the token is operand, add to output
   → If it is left paranthesis, push to opStack
   → If it is right paranthesis, pop the opstack until the corresponding left paranthesis is removed. add each operator at end of output str.
   → If the token is an operator (*, /, +, -) push it on the opStack. However, first remove any operator already on the opStack that have higher or equal precedence & add to ouptut.

3. When input string is completely processed, check if any operators in opStack that have higher or equal precedence & add to output

infix ⇒ $\boxed{A*B+C*D}$ A * B + C * D



postfix ⇒ $\boxed{AB*CD*+}$