Assignment 1

Answer1.a:

add(x): Insert element x in the priority queue.

deleteMin(): Delete the minimum element from the queue. And in priority queue, it's the first element.

Size(): Returns the size of the queue.

Implementation of above function using link list:

```
public class SinglyLinkedListForQueue{
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }

   public Node head = null;
   public Node tail = null;

   public void add(int data) {
      Node newNode = new Node(data);

      if(head == null) {
         head = newNode;
         tail = newNode;
      }
```

```
    else {
        if(tail.data<=data)
        {
            tail.next = newNode;
            tail = newNode;
        }
        else{
            Node current = head;
            Node temp=null;
            while(current!=null){
                if(current.data<data){
                    temp = current;
                    current=current.next;
                }
                else break;
            }

            if(temp==null){
                newNode.next=current;
                head=newNode;
            }else{
                temp.next = newNode;
                newNode.next = current;
            }
        }
    }
}

public void deleteMin(){
```

```
    head=head.next;
}



 public int size(){
    Node current = head;
    int count = 0;
    while(current!=null){
        count++;
        current=current.next;
    }
    return count;
}

public void display() {
    Node current = head;

    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    System.out.println("Nodes of singly linked list: ");
    while(current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
```

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

        SinglyLinkedListForQueue sl = new SinglyLinkedListForQueue();

        sl.add(7);
        sl.add(1);
        sl.add(2);
        sl.add(5);
        sl.add(9);
        sl.add(3);
        sl.add(4);

        sl.display();
        System.out.println("Total Element: "+ sl.size());
        sl.deleteMin();

        sl.display();
         System.out.println("Total Element after deleteMin(): "+ sl.size());
    }
}
```

Analysis if add(x):

Whenever we try to add an element in our priority queue using link list, we first check in which location it needs to be place because we have to implement priority queue. And as the by default priority is ascending order hence, we tried to implement that only.

So, the worst case would be the when we need to add the element at the position one less than the end in our link list. And for that we need to traverse the entire queue.

Hence the time complexity would be O(n).

Analysis if deleteMin(x):

Since, in our case, the queue is already sorted and this means the first element is the minimum element. Hence this function just deletes the first element by incrementing the head position Giving a constant time complexity O(1).

Analysis if size(x):

Since, we are traversing the entire queue (in our case is list) to get the size. Hence the time complexity is always O(n).

Answer1.b:

Implementation of stack push(x) and pop() using two queue:

```
import java.util.LinkedList;
import java.util.Queue;

public class StackUsingQueue {
   Queue<Integer> q1= new LinkedList<>();
   Queue<Integer> q2= new LinkedList<>();

   public void push(Integer val){
      q1.add(val);
   }

   public void pop(){
      int size = q1.size();
      while(size>0)
      {
         q2.add(q1.remove());
         size--;
```

```
        }
        q2.remove();
        size = q2.size();
        while(size>0)
        {
            q1.add(q2.remove());
            size--;

        }

    }

    public void Display(){
        System.out.println(q1);
    }
public static void main(String[] args)
{

    StackUsingQueue st = new StackUsingQueue();
    st.push(1);
    st.push(3);
    st.push(6);
    st.push(6);
    st.push(2);

    st.Display();
    st.pop();
    st.Display();
    st.pop();
```

```
    st.Display();

}
}
```

Answer2.a: Swap two adjacent element in singly linked list

```
public class SinglyLinkedList {
    class Node{
        int data;
        Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public Node head = null;
    public Node tail = null;

    public void addNode(int data) {
        Node newNode = new Node(data);

        if(head == null) {
            head = newNode;
            tail = newNode;
        }
```

```
    else {
        tail.next = newNode;
        tail = newNode;
    }
}

public void display() {
    Node curr = head;

    if(head == null) {
        System.out.println("Empty List");
        return;
    }
    System.out.println("Nodes: ");
    while(curr != null) {
        System.out.print(curr.data + " ");
        curr = curr.next;
    }
    System.out.println();
}

public int size(){
    Node curr = head;
    int count = 0;
    while(curr!=null){
        count++;
        curr=curr.next;
    }
    return count;
```

```
}

public void swap(int pos,int next_prev){
   if(size()==0){
      System.out.println("Empty List");
      return;
   }

   if(size()<pos){
      System.out.println("No Element is present in position: "+pos);
      return;
   }
   else if(next_prev==1){
      if(size()==pos)
         System.out.println("No Next adjacent Element is present after Positon: "+pos);
      else if(size()==2){
            Node curr=head;
            tail=head;
            head=curr.next;
            head.next=tail;
            tail.next=null;
            return;
      }
      else if(pos==1){
         Node temp = head.next;
         head.next=head.next.next;
         temp.next=head;
         head=temp;
      }
```

```
        else{
            Node curr = head;
            int currpos=1;
            while(currpos!=pos-1){
                curr=curr.next;
                currpos++;
            }

            Node temp;
            temp = curr.next;
            curr.next = temp.next;
            temp.next=curr.next.next;
            curr.next.next=temp;
            return;
        }
    }
    else if(next_prev==0){
        if(1==pos)
            System.out.println("No Previous adjacent Element is present before Positon: "+pos);
        else if(size()==2){
                Node curr=head;
                tail=head;
                head=curr.next;
                head.next=tail;
                tail.next=null;
return;
        }
        else if(pos==2){
            Node temp = head.next;
```

```
            head.next=head.next.next;
            temp.next=head;
            head=temp;
return;
        }
        else{
            Node curr = head;
            int currpos=1;
            while(currpos!=pos-2){
                curr=curr.next;
                currpos++;
            }

            Node temp;
            temp = curr.next;
            curr.next = temp.next;
            temp.next=curr.next.next;
            curr.next.next=temp;
            return;
        }
    }
}

public static void main(String[] args) {

    SinglyLinkedList sl = new SinglyLinkedList();

    sl.addNode(1);
    sl.addNode(2);
```

```
        sl.addNode(3);
        sl.addNode(4);

        sl.display();

        sl.swap(1,0); //Swap funtion takes the position where we want to swap and next_prev value(0/1),
        //0 for swap with Previous element and 1 for next Element
                // System.out.println(sList.size());

        sl.display();

        sl.swap(3,0);

        sl.display();

    }
}
```

Answer2.a: Swap two adjacent element in Doubly linked list

```
public class DoublyLinkedList {

    class Node{
        int data;
        Node previous;
        Node next;

        public Node(int data) {
            this.data = data;
        }
```

```
}

Node head, tail = null;

public void add(int data) {
   Node newNode = new Node(data);

   if(head == null) {
      head = tail = newNode;
      head.previous = null;
      tail.next = null;
   }
   else {
      tail.next = newNode;
      newNode.previous = tail;
      tail = newNode;
      tail.next = null;
   }
}

public void display() {
   Node curr = head;
   if(head == null) {
      System.out.println("Empty List");
      return;
   }
   System.out.println("Nodes: ");
   while(curr != null) {
```

```
        System.out.print(curr.data + " ");
        curr = curr.next;
    }
     System.out.println();
}

public int size(){
    Node curr = head;
    int count=0;
    while(curr!=null){
        count++;
        curr=curr.next;
    }
    return count;
}

public void swap(int pos,int next_prev){
    if(size()==0){
        System.out.println("Empty List");
        return;
    }


    if(size()<pos){
        System.out.println("No Element is present in position: "+pos);
        return;
    }

    else if(next_prev==1){
```

```
if(size()==pos)
   System.out.println("No Next adjuscent Element is present after Positon: "+pos);
else if(size()==2){
       head=tail;
       tail = head.previous;
       head.next=head.previous;
       head.previous=null;
       tail.previous=tail.next;
       tail.next=null;
       return;
}
else if(pos==1){
   Node temp = head.next;
   head.next=head.next.next;
   temp.next.previous=head;
   head.previous = temp;
   temp.next = head;
   temp.previous = null;
   head = temp;
   return;
}
else{
   Node curr = head;
   int currpos=1;
   while(currpos!=pos){
      curr=curr.next;
      currpos++;
   }
```

```
        Node temp = curr.next;
        if(temp.next!=null){
        curr.next=temp.next;
        curr.next.previous=curr;
        curr.previous.next = temp;
        temp.previous = curr.previous;
        temp.next = curr;
        curr.previous = temp;
           return;
        }
        else{
           curr.previous.next=temp;
           temp.previous=curr.previous;
           temp.next=curr;
           curr.next=null;
           curr.previous=temp;
           tail=curr;
           return;
        }
     }
  }

  else if(next_prev==0){
     if(1==pos)
        System.out.println("No Previous adjuscent Element is present before Positon: "+pos);
     else if(size()==2){
           head=tail;
           tail = head.previous;
           head.next=head.previous;
```

```
        head.previous=null;
        tail.previous=tail.next;
        tail.next=null;
        return;
    }
    else if(pos==size()){
       Node temp = tail.previous;
       tail.previous=tail.previous.previous;
       temp.previous.next=tail;
       tail.next = temp;
       temp.previous = tail;
       temp.next = null;
       tail = temp;
       return;
    }
    else{
       Node curr = head;
       int currpos=1;
       while(currpos!=pos){
          curr=curr.next;
          currpos++;
       }

       Node temp = curr.previous;
       if(temp.previous!=null){
       curr.previous=temp.previous;
       curr.previous.next=curr;
       curr.next.previous = temp;
       temp.next = curr.next;
```

```
            temp.previous = curr;
            curr.next = temp;
               return;
            }
            else{
               temp.next=curr.next;
               curr.next.previous=temp;
               curr.previous=null;
               curr.next=temp;
               temp.previous=curr;
               head=curr;
               return;
            }
         }
      }


}

public static void main(String[] args) {

   DoublyLinkedList dl = new DoublyLinkedList();
   dl.add(1);
   dl.add(2);
   dl.add(3);
   dl.add(4);
   dl.add(5);

   dl.display();
```

```
        dl.swap(1,0);//Swap funtion takes the position where we want to swap and next_prev value(0/1),
        //0 for swap with Previous element and 1 for next Element

        dl.display();

        dl.swap(3,0);

        dl.display();
    }
}
```

Answer 5: Reverse the Element in the Doubly Linked list

```
public class DoublyLinkedList {

    class Node{
        int data;
        Node previous;
        Node next;

        public Node(int data) {
            this.data = data;
        }
    }

    Node head, tail = null;

    public void add(int data) {
        Node newNode = new Node(data);
```

```
    if(head == null) {
        head = tail = newNode;
        head.previous = null;
        tail.next = null;
    }
    else {
        tail.next = newNode;
        newNode.previous = tail;
        tail = newNode;
        tail.next = null;
    }
}

public void display() {
    Node curr = head;
    if(head == null) {
        System.out.println("Empty List");
        return;
    }
    System.out.println("Nodes: ");
    while(curr != null) {

        System.out.print(curr.data + " ");
        curr = curr.next;
    }
     System.out.println();
}
```

```
public int size(){
    Node curr = head;
    int count=0;
    while(curr!=null){
        count++;
        curr=curr.next;
    }
    return count;
}

public void reverseMade(){
    Node curr = head;
    Node tempStore = new Node(0);

    if(head==null){
        System.out.println("Empty List");
        return;
    }

    head =tail;
    tail = curr;
    while(curr != null) {
        tempStore = curr.next;
        curr.next = curr.previous;
        curr.previous=tempStore;
        curr=curr.previous;
    }
}
```

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

        DoublyLinkedList dl = new DoublyLinkedList();
        dl.add(1);
        dl.add(2);
        dl.add(3);
        dl.add(4);
        dl.add(5);

        dl.display();

        dl.reverseMade();

        dl.display();
    }
}
```

Answer 6: Implement the MinStack

```
import java.util.*;
class MinStack
{
Stack<Integer> s;
Integer minEle;

MinStack() { s = new Stack<Integer>(); }

    public void display(){
        System.out.println(s);
    }
```

```
public void getMin()
{
if (s.isEmpty())
System.out.println("Empty Stack");
else
System.out.println("Minimum Element: " + minEle);
}

public void getTop()
{
if (s.isEmpty())
{
System.out.println("Stack is empty ");
return;
}

Integer val = s.peek();

System.out.print("Top Element: ");



if (val < minEle)
System.out.println(minEle);
else
System.out.println(val);
}
```

```
public void pop()
{
if (s.isEmpty())
{
System.out.println("Stack is empty");
return;
}

Integer val = s.pop();

if (minEle>val)
{
System.out.println(minEle);
minEle = 2*minEle - val;
}

else
System.out.println(val);
}

public void push(Integer val)
{
if (s.isEmpty())
{
minEle = val;
s.push(val);
return;
}
```

```
if (minEle>val)
{
s.push(2*val - minEle);
minEle = val;
}

else
s.push(val);
}



public static void main(String[] args)
{
MinStack s = new MinStack();
s.push(3);
s.push(5);
s.getMin();
s.push(2);
s.push(1);
s.getMin();
s.pop();
s.getMin();
s.pop();
s.getTop();
      s.display();
}
}
```