“Doubly Linked List”.

1. Which of the following is false about a doubly linked list?
a) We can navigate in both the directions
b) It requires more space than a singly linked list
c) The insertion and deletion of a node take a bit longer
d) Implementing a doubly linked list is easier than singly linked list
View Answer
Answer: d
Explanation: A doubly linked list has two pointers ‘left’ and ‘right’ which enable it to traverse in either direction. Compared to singly liked list which has only a ‘next’ pointer, doubly linked list requires extra space to store this extra pointer. Every insertion and deletion requires manipulation of two pointers, hence it takes a bit longer time. Implementing doubly linked list involves setting both left and right pointers to correct nodes and takes more time than singly linked list.

2. Given the Node class implementation, select one of the following that correctly inserts a node at the tail of the list.

```
public class Node
{
          protected int data;
          protected Node prev;
          protected Node next;
          public Node(int data)
          {
                    this.data = data;
                    prev = null;
                    next = null;
          }
          public Node(int data, Node prev, Node next)
          {
                    this.data = data;
                    this.prev = prev;
                    this.next = next;
          }
          public int getData()
          {
                    return data;
          }
          public void setData(int data)
          {
```

```
                this.data = data;
        }
        public Node getPrev()
        {
                return prev;
        }
        public void setPrev(Node prev)
        {
                this.prev = prev;
        }
        public Node getNext
        {
                return next;
        }
        public void setNext(Node next)
        {
                this.next = next;
        }
}
public class DLL
{
        protected Node head;
        protected Node tail;
        int length;
        public DLL()
        {
                head = new Node(Integer.MIN_VALUE,null,null);
                tail = new Node(Integer.MIN_VALUE,null,null);
                head.setNext(tail);
                length = 0;
        }
}
```

a)


```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().setNext(node);
        tail.setPrev(node);
```

```
        length++;
}
```

b)

Take Data Structure I Practice Tests - Chapterwise!
Start the Test Now: Chapter 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().getPrev().setNext(node);
        tail.setPrev(node);
        length++;
}
```

c)

```
public void insertRear(int data)
{
        Node node = new Node(data,tail.getPrev(),tail);
        node.getPrev().setNext(tail);
        tail.setPrev(node);
        length++;
}
```

d)

```
public void insertRear(int data)
{
        Node node = new Node(data,head,tail);
        node.getPrev().setNext(node);
        tail.setPrev(node);
        length++;
}
```

View Answer

Answer: a

Explanation: First create a new node whose ‘prev’ points to the node pointed to by the ‘prev’ of tail. The ‘next’ of the new node should point to tail. Set the ‘prev’ of tail to point to new node and the ‘prev’ of new node to point to the new node.

3. What is a memory efficient double linked list?
a) Each node has only one pointer to traverse the list back and forth
b) The list has breakpoints for faster traversal
c) An auxiliary singly linked list acts as a helper list to traverse through the doubly linked list
d) A doubly linked list that uses bitwise AND operator for storing addresses
View Answer
Answer: a
Explanation: Memory efficient doubly linked list has only one pointer to traverse the list back and forth. The implementation is based on pointer difference. It uses bitwise XOR operator to store the front and rear pointer addresses. Instead of storing actual memory address, every node store the XOR address of previous and next nodes.

4. Which of the following piece of code removes the node from a given position?
a)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                        return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
               else
               {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext();
                }
                temp.getNext().setPrev(temp.getPrev());
                temp.getPrev().setNext(temp.getNext());
        }
```

```
        size--;
}
```

b)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext();
                }
                temp.getNext().setPrev(temp.getNext());
                temp.getPrev().setNext(temp.getPrev());
        }
        size--;
}
```

c)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
```

```
        else
        {
                if(head == null)
                        return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext().getNext();
                }
                temp.getNext().setPrev(temp.getPrev());
                temp.getPrev().setNext(temp.getNext());
        }
        size--;
}
```

d)

```
public void remove(int pos)
{
        if(pos<0 || pos>=size)
        {
                System.out.println("Invalid position");
                return;
        }
        else
        {
                if(head == null)
                        return;
                if(pos == 0)
                {
                        head = head.getNext();
                        if(head == null)
                        tail = null;
                }
                else
                {
```

```
                        Node temp = head;
                        for(int i=1; i<position; i++)
                        temp = temp.getNext().getNext();
                }
                temp.getNext().setPrev(temp.getNext());
                temp.getPrev().setNext(temp.getPrev());
        }
        size--;
}
```

View Answer

Answer: a

Explanation: If the position to be deleted is not the head, advance to the given position and manipulate the previous and next pointers of next and previous nodes respectively.

5. How do you calculate the pointer difference in a memory efficient double linked list?

a) head xor tail

b) pointer to previous node xor pointer to next node

c) pointer to previous node – pointer to next node

d) pointer to next node – pointer to previous node

View Answer

Answer: b

Explanation: The pointer difference is calculated by taking XOR of pointer to previous node and pointer to the next node.

6. What is the worst case time complexity of inserting a node in a doubly linked list?

a) O(nlogn)

b) O(logn)

c) O(n)

d) O(1)

View Answer

Answer: c

Explanation: In the worst case, the position to be inserted maybe at the end of the list, hence you have to traverse through the entire list to get to the correct position, hence O(n).

7. How do you insert a node at the beginning of the list?

a)

```
public class insertFront(int data)
```

```
{
        Node node = new Node(data, head, head.getNext());
        node.getNext().setPrev(node);
        head.setNext(node);
        size++;
}
```

b)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head);
        node.getNext().setPrev(node);
        head.setNext(node);
        size++;
}
```

c)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head.getNext());
        node.getNext().setPrev(head);
        head.setNext(node);
        size++;
}
```

d)

```
public class insertFront(int data)
{
        Node node = new Node(data, head, head.getNext());
        node.getNext().setPrev(node);
        head.setNext(node.getNext());
        size++;
}
```

View Answer

Answer: a

Explanation: The new node's previous pointer will point to head and next pointer will point to the current next of head.

8. Consider the following doubly linked list: head-1-2-3-4-5-tail. What will be the list after performing the given sequence of operations?

```
        Node temp = new Node(6,head,head.getNext());
        Node temp1 = new Node(0,tail.getPrev(),tail);
        head.setNext(temp);
        temp.getNext().setPrev(temp);
        tail.setPrev(temp1);
        temp1.getPrev().setNext(temp1);
```

a) head-0-1-2-3-4-5-6-tail
b) head-1-2-3-4-5-6-tail
c) head-6-1-2-3-4-5-0-tail
d) head-0-1-2-3-4-5-tail

View Answer

Answer: c
Explanation: The given sequence of operations performs addition of nodes at the head and tail of the list.

9. What is the functionality of the following piece of code?

```
public int function()
{
        Node temp = tail.getPrev();
        tail.setPrev(temp.getPrev());
        temp.getPrev().setNext(tail);
        size--;
        return temp.getItem();
}
```

a) Return the element at the tail of the list but do not remove it
b) Return the element at the tail of the list and remove it from the list
c) Return the last but one element from the list but do not remove it
d) Return the last but one element at the tail of the list and remove it from the list

View Answer

Answer: b
Explanation: The previous and next pointers of the tail and the last but one element are manipulated, this suggests that the last node is being removed from the list.

10. Consider the following doubly linked list: head-1-2-3-4-5-tail. What will be the list after performing the given sequence of operations?

```
        Node temp = new Node(6,head,head.getNext());
```

```
        head.setNext(temp);
        temp.getNext().setPrev(temp);
        Node temp1 = tail.getPrev();
        tail.setPrev(temp1.getPrev());
        temp1.getPrev().setNext(tail);
```

a) head-6-1-2-3-4-5-tail
b) head-6-1-2-3-4-tail
c) head-1-2-3-4-5-6-tail
d) head-1-2-3-4-5-tail

View Answer

Answer: b

Explanation: A new node is added to the head of the list and a node is deleted from the tail end of the list.