[1] Creating an array:

```
Foo[ ] x = new foo[100];

Foo[y] = x;
```

[2] Iterating over the elements of an array:

```
for(int i=0; i < x.length; i++)
{
    System.out.println (x[i]);
}
```

[3] Copying an array:

```
public static Object[] copyArray(Object[ ] source)
{
    Object[ ] copy = new Object [source.length];
    for (int i=0; i< source.length ; i++)
    {
        copy[i] = source[i];
    }
    return copy;
}
```

[4] Resizing an array:

```
static Object [ ] resize (Object[ ] oldArray, int newCapacity)
{
    Object[ ] newArray = new Object [newCapacity];
    for (int i=0; i< oldArray.length; i++)
    {
        newArray[i] = oldArray[i];
    }
    return newArray;
}
```

5 Reversing an array:

```
public static void reverse (Object[] array)
{
    Object[] tempArray = new Object[array.length];
    int i = 0;
    int j = tempArray.length - 1;
    while (i < array.length)
    {
        tempArray[j] = array[i];
        i++;
        j--;
    }
    for (i=0; i < array.length; i++)
    {
        array[i] = tempArray[i];
    }
}
```

* Another method → in place:

```
public void reverse (Object[] array)
{
    int i=0;
    int j = array.length - 1;
    while (i<j)
    {
        Object temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        i++;
        j--;
    }
}
```

[6] Shifting an array left:

```
public static void shiftleft (Object array[])
{
    for(int i=1 ; i< array.length; i++)
    {
        array[i-1] = array[i];
    }
    array [a.length-1] =null ;
}
```

[7] Shifting an array right:

```
public static void shiftRight (Object array[])
{
    for (i=array.length -1; i>0 ; i--)
    {
        array [i] = array [i-1];
    }
    array[0]=null;
}
```

[8] Inserting an element into an array:

```
public static void insert (Object[] array, int size, Object elem,
                           int index)
{
    if (size == array.length)
    {
        System.out.println("no space left");
    }
    else
    {
        for (i=index ; i<size ; i++)
        {
            array [i+1] =array [i];
        }
        array [index] =elem;
    }
}
```

Correction in All Notes

```
for (i=size; i>index;i++)
{   array[i] =array[i-1];
```

9 Removing an element from an array:

```
public static void remove (Object[] array, int size, int index)
{
    for (i= index+1 ; i<size ; i++)
    {
        array[i-1] =array[i];
    }
    array[size-1] =null;
}
```

10 Rotating an array. Left:

```
public static void rotateLeft (Object array[])
{
    Object firstElement =array[0];
    for (i=1 ; i< array.length ; i++)
    {
        array[i-1] =array[i];
    }
    array[a.length-1] = firstElement;
}
```

11 Rotating an array right:

```
public static void rotateRight (Object array[])
{
    Object lastElement =array[array.length-1];
    for(i=array.length-1 ; i>0 ; i--)
    {
        array[i] =array[i-1];
    }
    array[0] =lastElement;
}
```

Circular Array:

① Iteration:

* Forward:

```
int k = start;
for (int i=0; i<size; i++)
{
    System.out.println (Array[k]);
    k=(k+1) % array.length;
}
```

next available position = (start+size) % array.length.

~~next available position~~ – (~~start+size~~) ~~%~~ ~~array.length~~

* Backward:

```
int k = (start+size-1) % array.length
for (i=0; i<size; i++)
{
    System.out.println (array[k]);
    k--;
    if (k == -1)
    {
        k = array length - 1;
    }
}
```

② Linearizing a circular array → Allnotes → pdf page 10

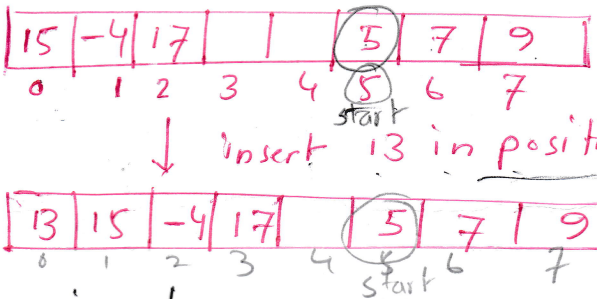
③ Resizing " " " → " " " 10-11

④ Inserting an element in a circular array:

```
public static int insert(Object[] cirArray, int start, int size,
                         Object element, int position)
{
    if (size == cirArray.length)
    {
        resize (cirArray, start, size, size * 2);
    }
    int nShifts = size - position;
    int from = (start + size - 1) % cirArray.length;
    int to = (from+1) % cirArray.length;
    for (i=0; i< nShifts ; i++)
    {
        cirArray [to] = cirArray [from];
        to = from;
        from --;
        if (from == -1)
            from == cirArray.length - 1;
    }
```

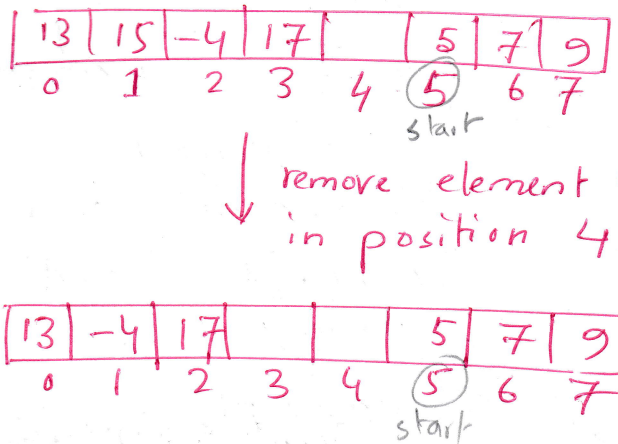
P.T.O

```
int index = (start + position) % cirArray.length;
cirArray[index] = element;
return index;
}
```

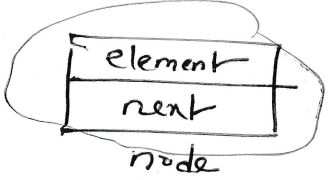


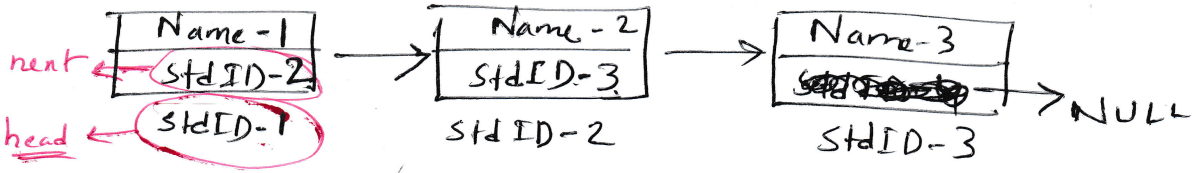
5 Removing an element in a circular array:

```
public static Object remove(Object[] cirArray, int start, int size,
                            Object element, int position)
{
    int index = (start + position) % cirArray.length;
    Object removed = cirArray[index];
    int nShifts = size - position - 1;
    int to = index;
    int from = (to + 1) % cirArray.length;
    for (i = 0; i < nShifts; i++)
    {
        cirArray[to] = cirArray[from];
        to = from;
        from = (from + 1) % cirArray.length;
    }
    cirArray[from] = null;
    return removed;
}
```



Singly Link List :






```
public class Node
{
    public Object element;
    public Node next;
    public Node(Object e, Node n)
    {
        element = e;
        next = n;
    }
}
```

1. Creating a list of elements :

```
Node head = null;
head = new Node("Hello", null);
```

2. Iterating over the elements :

```
for(Node n = head; n != null; n = n.next)
{
    System.out.println(n.element);
}
```

3. Counting the number of elements in a List :

```
public static int count(Node head)
{
    int count = 0;
    for(Node n = head; n != null; n = n.next)
    {
        count++;
    }
    return count;
}
```

4 Getting an element given the index into a list:

```
public static Object get (Node head, int index)
{
    if (index<0)
    {
        return null;
    }
    int i=0;
    for (Node n=head; n!=null; n=n.next)
    {
        if (i==index)
        {
            return n.element;
        }
        else
        {
            i++;
        }
    }
    return null;
}
```

if size given

```
public static Object get (Node head, int size, int index)
{
    if (index<0 || index>=size)
    {
        return null;
    }
    Node n=head;
    for (i=0; i<index; i++, n=n.next)
    {
    }
    return n.element;
}
```

4 Continue: Returns the node at the given index in the given list

```
public static Node nodeAt (Node head, int size, int index)
{   if (index<0 || index>=size)
    {   return null;
    }
    Node n=head;
    for (i=0; i<index; i++, n=n.next) ;
    return n;
}
```

5 Setting an element given the index into a List:

```
public static Object get (Node head, int size, int index, Object elem)
{   Node node = nodeAt (head, size, index);
    if (node==null)
    {   return null;
    }
    else
    {   Object oldElement = node.element;
        node.element = elem;
        return oldElem;
    }
}
```

6 Searching for an element in a list:

```
public static int indexOf (Node head, int size, Object element)
{
    int i = 0;
    for (Node n = head; n != null; n = n.next, i++)
    {
        if (n.element.equals (element))
        {
            return i;
        }
    }
    return -1;
}
```

7 Inserting an element into a list:

```
public static Node insert (Node head, int size, Object element, int index)
{
    if (index < 0 || index > size)
    {
        System.out.println("Invalid index");
    }
    Node newNode = new Node (element, null);
    if (index == 0)
    {
        newNode.next = head;
        head = newNode;
    }
    else
    {
        Node pred = nodeAt (index - 1);
        newNode.next = pred.next;
        pred.next = newNode;
    }
    return head;
}
```

[8] Removing a element from the list:

```
public static Node remove (Node head, int size, int index)
{
    if (index < 0 || index >= size)
    {
        System.out.println ("invalid index");
    }
    Node removedNode = null;
    if (index == 0)
    {
        removedNode = head;
        head = head.next;
    }
    else
    {
        Node pred = nodeAt (index-1);
        removedNode = pred.next;
        pred.next = removedNode.next;
    }
    removedNode.element = null;
    removedNode.next = null;
    return head;
}
```

9 Copying a list:

```
public static Node copyList(Node source)
{
    Node copyHead = null;
    Node copyTail = null;
    for (Node n = source; n != null; n = n.next)
    {
        Node newNode = new Node(n.element, null);
        if (copyHead == null)
        {
            copyHead = newNode;
            copyTail = copyHead;
        }
        else
        {
            copyTail.next = newNode;
            copyTail = copyTail.next;
        }
    }
    return copyHead;
}
```

10 Rotating a list left:

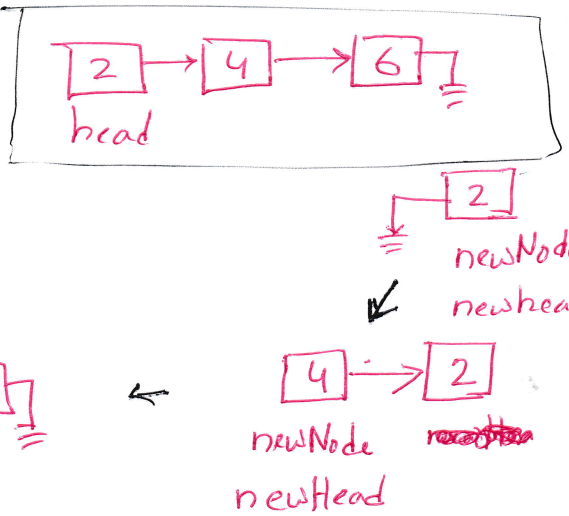
```
public static Node rotateLeft (Node head, int size)
{
        Node oldHead = head;
        head = head.next;
        Node temp = head;
        while (temp.next != null)
        {
            temp = temp.next;
        }
        temp.next = oldHead;
        oldHead.next = null;
        return head;
}
```

11 Rotating a list right:

```
public static Node rotateRight (Node head, int size)
{
        Node p = null;
        Node q = head;
        while (q.next != null)
        {
            p = q        → [error in All Notes. Do the correction]
            q = q.next;
        }
        q.next = head;
        head = q;
        p.next = null;
        return head;
}
```

Singly Link List

```
Node reverse (Node head) ;
{
    Node & newHead = null;
    for(Node n=head; n!=null; n=n.next)
    {
        Node newNode = new Node (n.element, null);
        newNode.next = newHead;
        newHead = newNode;
    }
    return newHead;
}
```

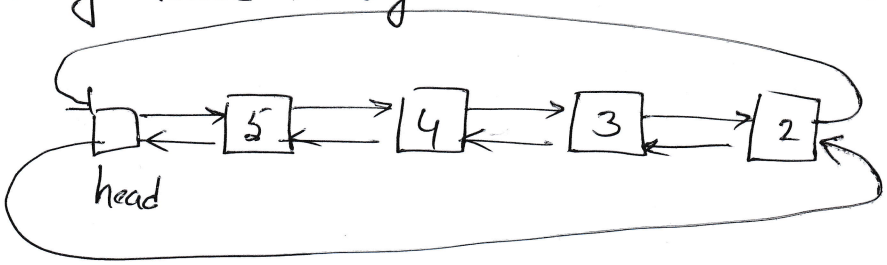


Practice:

* iteration
* count
* find an element given the index of a list
* update " " " " " " " " ".
* insert an element/node into a list.
* remove " " " " " " ".
* copy a list.
* reverse a list.
* rotatate right/left for one-time/multiple times.
* Sort a list.

Doubly Circular List with Dummy Head

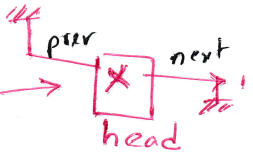
Dummy Headed Doubly Linked Circular Lists:

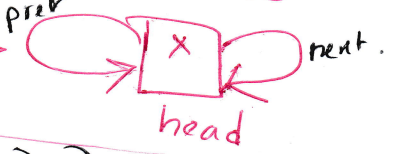



```
public class Node
{
    public Object element;
    public Node next;
    public Node prev;
    public Node (Object e, Node n, Node p)
    {
        element = e;
        next = n;
        prev = p;
    }
}
```

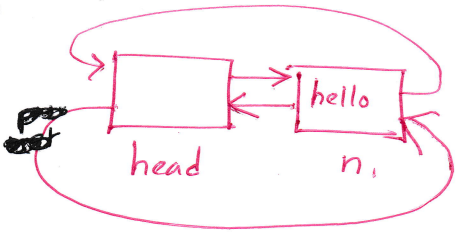
Creating a list of elements:

```
Node head = new Node (null, null, null);
head.next = head.prev = head;
```



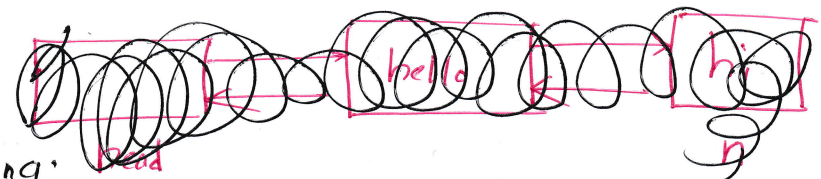



```
Node n = new Node ("hello", null, null);
n.next = head.next;
n.prev = head;
head.next = n;
n.next.prev = n;
```



Inserting a node "hello", after head.

Again Traversing:

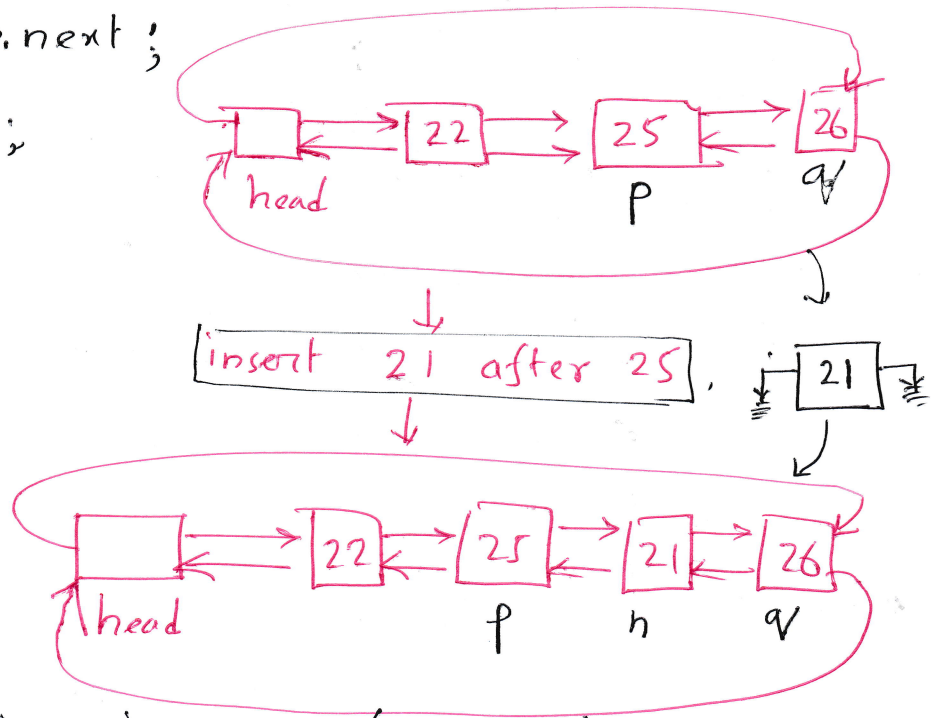


Iterating / Traversing:

Same as singly Link list.

Inserting an element into a list:

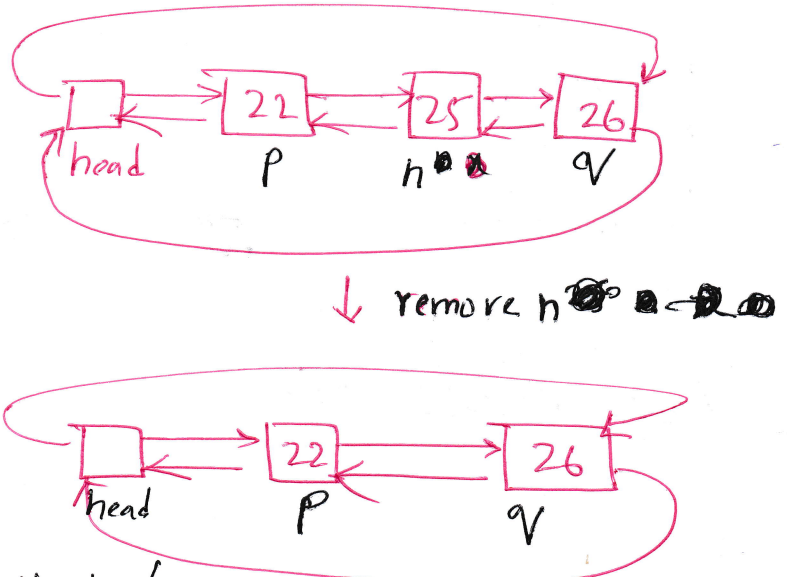
```
public static Node insertAfter (Node p, Object elem)
{
    Node n = new Node(elem, null, null);
    Node q = p.next;
    n.next = q;
    n.prev = p;
    p.next = n;
    q.prev = n;
    return n;
}
```



* Insert into first position. ⟶ insertAfter (head, elem)
* Insert into last position ⟶ insertAfter (head.prev, elem)

Remove an element from a list:

```
public static void removeNode (Node n)
{
    Node p = n.prev;
    Node q = n.next;
    p.next = q;
    q.prev = p;
    n.next = n.prev = null;
    n.element = null;
}
```



* remove first node : removeNode (head.next);
* remove last node : removeNode (head.prev);