MILESTONE 3

1. https://leetcode.com/problems/reorder-list/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public void reorderList(ListNode head) {
        ListNode cur=new ListNode(),c=new ListNode(),p=new ListNode(),prv=new
ListNode(),temp=new ListNode();
          int i,n=0;
          cur=head;
    temp=head;
        while(temp!=null)//finding the length
        {
            temp=temp.next;
            n++;
        }
        if(n>2){
    for(i=1;i<=n;i++)
          {
                    if(i%2==0)
                    {
                              prv=cur;
                              while(cur.next!=null)
                              {
                                        p=cur;
                                        cur=cur.next;
                              }
                              cur.next=prv;
```

```
                        p.next=null;
                        c.next=cur;
                   }
                   else
                   c=cur;
                   cur=cur.next;
              }
        }
        }
}
```

2. https://leetcode.com/problems/min-stack/

```
class MinStack {
    Stack<Integer> s=new Stack();
    Stack<Integer> min_values=new Stack();

    public MinStack() {

    }

    public void push(int val) {
        if(min_values.isEmpty() || val<=min_values.peek())
            min_values.push(val);
        s.push(val);
    }

    public void pop() {
        if(s.peek().equals(min_values.peek()))
            min_values.pop();
        s.pop();
    }

    public int top() {
        return s.peek();
    }

    public int getMin() {
        return min_values.peek();
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(val);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```

3. https://leetcode.com/problems/implement-stack-using-queues

```
class MyStack {
    Queue<Integer> q1=new LinkedList();
    Queue<Integer> q2=new LinkedList();
    int top;

    public MyStack() {

    }

    public void push(int x) {
        q2.add(x);
        top=x;
        while(!q1.isEmpty())
        {
            q2.add(q1.remove());
        }
        Queue<Integer> temp=q1;
        q1=q2;
        q2=temp;
    }

    public int pop() {
        int t=q1.remove();
        if(!q1.isEmpty())
        {
            top=q1.peek();
        }
        return t;
    }

    public int top() {
        return q1.peek();
    }

    public boolean empty() {
        return q1.isEmpty();
    }
}

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack obj = new MyStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * boolean param_4 = obj.empty();
 */
```

4. https://leetcode.com/problems/implement-queue-using-stacks

```
class MyQueue {
Stack<Integer> s1=new Stack<>();
    Stack<Integer> s2=new Stack<>();
    public MyQueue() {

    }

    public void push(int x) {
        s1.push(x);
    }

    public int pop() {
        if(s2.isEmpty())
            shift();
        return s2.pop();
    }

    public int peek() {
        if(s2.isEmpty())
            shift();
        return s2.peek();
    }

    public boolean empty() {
        boolean ans=s1.isEmpty() && s2.isEmpty();
        return ans;
    }

    public void shift(){
        while(!s1.isEmpty())
        {
            int t=s1.pop();
            s2.push(t);
        }
    }
}

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * boolean param_4 = obj.empty();
 */
```

5. https://leetcode.com/problems/merge-two-sorted-lists

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
```

```
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(0);
        ListNode p=head;
        ListNode p1=l1;
        ListNode p2=l2;
        while(p1!=null && p2!=null){
        if(p1.val < p2.val){
            p.next=p1;
            p1=p1.next;
        }
        else{
            p.next=p2;
            p2=p2.next;
        }
        p=p.next;
    }

    if(p1!=null){
        p.next = p1;
    }

    if(p2!=null){
        p.next = p2;
    }

    return head.next;
    }
}
```

6. https://leetcode.com/problems/linked-list-cycle

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
```

```
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head==null)
            return false;
        ListNode p1=head;
        ListNode p2=head.next;
        while(p1!=p2)
        {
            if(p2==null || p2.next==null)
                return false;
            p1=p1.next;
            p2=p2.next.next;
        }
        return true;
    }
}
```

7. https://leetcode.com/problems/subarray-product-less-than-k

```
class Solution {
    public int numSubarrayProductLessThanK(int[] nums, int k) {
        if(k<=1)
            return 0;
        int p,ans,l,r;
        p=1;
        ans=0;
        l=r=0;
        while(r<nums.length)
        {
            p=p*nums[r];
            while(p>=k)
            {
                p=p/nums[l];
                l++;
            }
            ans+=r-l+1;
            r++;
        }
        return ans;
    }
}
```