Notice

- (**NEW**) If a question doesn't ask for a specific running time, your implementation should provide the best. Otherwise you get only partial marks.
- Follow Java Code Conventions; otherwise you cannot get full points. The requirement on naming (section 4) and indentation (section 9) are hard. Other parts are suggestive.
- Make sure your codes can be compiled by command line. If you've problems, this and this web pages may help, and you're welcome to ask the graders.
- For each question, you submission should contain a .java source file and a screenshot of its execution. If you don't know how to take a screenshot, this website may help.
- Submit on learn@polyu, no hard-copies are required.

1. (30 points) Finish the class `ListOnArray` in Lab 5, implementing all the methods.

2. (25 points)
```
public class CustomerNode {
    String name;
    CustomerNode next;

    CustomerNode (String name) {
        this.name = name;
        next = null;
    }

    public String toString() {
        return name;
    }
}
```

   Write a class `CustomerList` that stores a sequence of customers (`CustomerNode` is given above) as a linked list. Besides the normal operations of a linked list, your class should provide a method that reverses the list: If the original list has customers $1, 2, 3, 4$, then the new list would be $4, 3, 2, 1$.

   You should use the original nodes, so after the method is called the original list is no longer available. Your method should run in $O(n)$ time and use $O(1)$ extra space (not counting the list itself).

3. (20 points) Write a class `CustomerQueue` that represents a queue of customers waiting as a line at a cashier. It should use the class `CustomerList` you've just written as the internal storage, and provide operations `enqueue`, `dequeue`, and `isEmpty` in $O(1)$ time.
   (Hint: Does your class `CustomerList` need a `tail`?)

4. (25 points) Add the following method to `CustomerQueue`:

   ```
   public CustomerQueue[] split(int k)
   ```

   It splits the queue into k queues of almost equal sizes, such that the first customer goes to the first queue, the second customer goes to the second queue, and so on. It should run in $O(n)$ time, where n is the number of customers waiting in the queue, and use $O(1)$ extra space (not counting the list itself).

   You may consider this scenario. Eleven students are waiting to pay at the only cashier of the cafeteria, while other two cashiers start working. The queue will immediately dissolve into three queues (what are their sizes?).

5. (Challenging and voluntary) Write another reverse method for your class `CustomerList`, which makes a copy of the original nodes, so that after it returns, you have two independent lists. It should run in $O(n)$ time and use $O(1)$ extra space (not counting the lists themselves).