Assignment1

张旭东 12011923

1. The explanation of the method $int[]calculateInitCards(int[]destroyOrder)$

To calculate $initOrder$ from $destroyOrder$, the solution is shown as following steps.

Step1:

create a object $initialorder$, which has the method of $dequeue$ and $enqueue$. (the method of $dequeue$ can delete the first element of $initialorder$ and the method of $enqueue$ can add a element to the tail of $initialorder$).

```
public Item dequeue(){ //向表头删除元素
        Item item= first.item;
        first= first.next;
        if (isEmpty()){
            last=null;
        }
        N--;
        return item;

    }
```

```
public void enqueue(Item item){       //从表尾添加元素
        Node oldlast=last;
        last=new Node();
        last.item=item;
        last.next=null;
        if (isEmpty()){
            first=last;
        }
        else {
            oldlast.next=last;
        }
        N++;
    }
```

Step2:

According to common sense, $destroyOrder[deatroyOrder.length-1]$ and $destroyOrder[deatroyOrder.length-2]$is the cards which is destroyed at the last. So, $initialorder.enqueue(destroyOrder[deatroyOrder.length-1])$ and then $initialorder.enqueue(destroyOrder[deatroyOrder.length-2])$. When the size of $initialorder$ is greater than 1, a variable $trans$ is created which is the element of $initialorder.dequeue$. Then, $trans$ is added to the tail of $initialorder$. After that, $destroyOrder[deatroyOrder.length-3]$ is also added to the tail of $initialorder$. The same action is repeated over and over until the first element of $destroteyorder$ is added to the tail of $initialorder$. At the time , a queue $initialorder$ whose order of elements is reverse to $initOrder$ we want is got. Last but not least, we reverse the order of element in $initialorder$ to get $initOrder$.

```
public static int[] calculateDestroyCards( int[] initOrder ) {
        // write your code here.

        QueueOfInitial <Integer> initialorder = new
QueueOfInitial<Integer>();
        for (int i =0;i<initOrder.length-1;i++){
            initialorder.enqueue(initOrder[i]);
        }
        int[] destroyorder=new int[initOrder.length];
        for (int i =0;i<destroyorder.length;i++){
            if (initialorder.size()>2){
                int de = initialorder.dequeue();
                destroyorder[i]=de;
```

```
                int de1 = initialorder.dequeue();
                initialorder.enqueue(de1);
            }
            else {
                destroyorder[i]=initialorder.dequeue();
            }
        }
        return destroyorder;
    }
```

Result:

From the following result, what can be known is that the method of $int[]calculateInitCards(int[]destroyOrder)$ is correct.

```
The initorder is:
1
2
3
4
The out is:
1
2
3
4
The method is correct

The initorder is:
1
3
2
4
The out is:
1
3
2
4
The method is correct

Process finished with exit code 0
```

```
for(int i =1;i<=2;++i){
            try {
                In fin1 =new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment1\\assignment
1\\assignment 1\\data\\"+i+".in");
                int[] arr1 = fin1.readAllInts();
                fin1.close();

                int[] destroyOrder = new int[arr1.length-1];
                for( int j = 0; j < destroyOrder.length; j ++ ){
                    destroyOrder[j] = arr1[j+1];
                }
                int[] initOrder = calculateInitCards(destroyOrder);

                In fin2 = new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment1\\assignment
1\\assignment 1\\data\\"+i+".out");
                int[] arr2= fin2.readAllInts();

                System.out.println("The initorder is:");
                for( int j : initOrder ){
                    System.out.println(j);
                }

                System.out.println("The out is:");
                for( int j : arr2 ){
                    System.out.println(j);
                }

                if (Arrays.equals(initOrder, arr2)){
                    System.out.println("The method is correct\n");
                }
                else {
                    System.out.println("The method isn't
correct\n");
                }

            }catch (IllegalArgumentException e){
                e.printStackTrace();
            }
```

```
        }
```

2.Test and proof

2.1 Test

The first step is to generate random new data. (The code is in the appendix). The number of cards is controlled by ourselves. Then run the program to get the result.

In the following case, the number of cards is 23. The result is following.

```
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2
23 13 9 22 14 12 20 21 5 15 11 1 8 6 17 7 10 18 3 19 16 4 2

Process finished with exit code 0
```

2.2 Proof

To prove its correctness, a method named $int[]calculateDestroyCards(int[]initOrder)$ is created. The method can calculate the $destroyedOrder$ using the $initOrder$ gotten by the method $int[]calculateInitCards(int[]destroyOrder)$. The code of the method is shown as below.

```
public static int[] calculateDestroyCards( int[] initOrder ) {
        // write your code here.

        QueueOfInitial <Integer> initialorder = new
QueueOfInitial<Integer>();
        for (int i =0;i<initOrder.length;i++){
            initialorder.enqueue(initOrder[i]);
        }
        int[] destroyorder=new int[initOrder.length];
        for (int i =0;i<destroyorder.length;i++){
```

```
            if (initialorder.size()>2){
                int de = initialorder.dequeue();
                destroyorder[i]=de;
                int de1 = initialorder.dequeue();
                initialorder.enqueue(de1);
            }
            else {
                destroyorder[i]=initialorder.dequeue();
            }
        }
        return destroyorder;
    }
```

The results are correct if the calculations are the same as the test data. The code of comparing test data and calculations is shown as below.

```
int[] destroy = calculateDestroyCards(initOrder);
                if (Arrays.equals(destroyOrder,destroy)){
                    System.out.println("The method is correct");
                }
                else {
                    System.out.println("The method isn't correct");
                }
```

In the following case, the number of cards is 100000. The compared result is shown as following.

```
The method is correct
The method is correct
The method is correct
The method is correct
The method is correct
The method is correct
The method is correct
The method is correct

Process finished with exit code 0
```

In a conclusion, the method of $int[]calculateInitCards(int[]destroyOrder)$ is correct.

Appendix

```
import java.util.Iterator;

public class QueueOfInitial <Item> implements Iterable<Item>{

    private Node first; //
    private Node last;
    private int N;


    private class Node{
        Item item;
        Node next;
    }

    public boolean isEmpty(){return first==null;}
    public int size(){return N;}

    public Item dequeue(){ //向表头删除元素
        Item item= first.item;
        first= first.next;
        if (isEmpty()){
            last=null;
        }
        N--;
        return item;

    }
    public void enqueue(Item item){       //从表尾添加元素
        Node oldlast=last;
        last=new Node();
        last.item=item;
        last.next=null;
        if (isEmpty()){
            first=last;
        }
        else {
            oldlast.next=last;
```

```
        }
        N++;
    }

    public Iterator<Item> iterator() {
        return new ListIterator();
    }

    private class ListIterator implements Iterator<Item>{
        private Node current=first;
        public boolean hasNext(){return current!=null;}
        public Item next(){
            Item item=current.item;
            current=current.next;
            return item;
        }
    }
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.File;
import java.util.Scanner;
import edu.princeton.cs.algs4.StdRandom;

/**
 * 细节：这里生成的数据并没有表示数组长度的元素,且生成的是destroyOrder
 */

public class GenData {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        int N= input.nextInt();
        int[] arr = new int[N];
        for( int i = 0, num =1; i < arr.length; ++i, num++ )
            arr[i] = num;
        StdRandom.shuffle(arr);
```

```
        new File("D:\\Study in SUSTech\\First semester of junior
year\\dsaaB\\lab\\assignment1\\assignment 1\\data\\").mkdirs();

        for( int i = 1; i <= 8; ++ i ) {
            try ( PrintWriter fout = new PrintWriter("D:\\Study in
SUSTech\\First semester of junior
year\\dsaaB\\lab\\assignment1\\assignment 1\\data\\"+i+".in") ) {
                for( int j = 0; j < i*1000 && j < arr.length; ++ j
)
                    fout.println(" " + arr[j]);
            } catch ( IOException e ) {
                e.printStackTrace();
            }
        }
    }
}
```

```
import edu.princeton.cs.algs4.In;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Scanner;

public class Cards {
    /**
     * Using the order the cards are destroyed, calculate the
initial order of the cards.
     * @param destroyOrder the order the cards are destroyed.
     * @return the initial order
     */
    public static int[] calculateInitCards( int[] destroyOrder ) {
        // write your code here.
        QueueOfInitial <Integer> initialorder = new
QueueOfInitial<Integer>();
        for(int i=destroyOrder.length-1;i>-1;i--){
            if (initialorder.size()>1){
                int trans= initialorder.dequeue();
                initialorder.enqueue(trans);
            }
```

```
            initialorder.enqueue(destroyOrder[i]);
        }
        int[] initial= new int[destroyOrder.length];

        for (int i=destroyOrder.length-1;i>-1;i--){
            initial[i] = initialorder.dequeue();
        }
        return initial;
    }

    public static int[] calculateDestroyCards( int[] initOrder ) {
        // write your code here.

        QueueOfInitial <Integer> initialorder = new
QueueOfInitial<Integer>();
        for (int i =0;i<initOrder.length;i++){
            initialorder.enqueue(initOrder[i]);
        }
        int[] destroyorder=new int[initOrder.length];
        for (int i =0;i<destroyorder.length;i++){
            if (initialorder.size()>2){
                int de = initialorder.dequeue();
                destroyorder[i]=de;
                int de1 = initialorder.dequeue();
                initialorder.enqueue(de1);
            }
            else {
                destroyorder[i]=initialorder.dequeue();
            }
        }
        return destroyorder;
    }


    public static void main( String[] args ) {

        for(int i =1;i<=2;++i){
            try {
                In fin1 =new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment1\\assignment
1\\assignment 1\\data\\"+i+".in");
                int[] arr1 = fin1.readAllInts();
```

```
                fin1.close();

                int[] destroyOrder = new int[arr1.length-1];
                for( int j = 0; j < destroyOrder.length; j ++ ){
                    destroyOrder[j] = arr1[j+1];
                }
                int[] initOrder = calculateInitCards(destroyOrder);

                In fin2 = new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment1\\assignment
1\\assignment 1\\data\\"+i+".out");
                int[] arr2= fin2.readAllInts();

                System.out.println("The initorder is:");
                for( int j : initOrder ){
                    System.out.println(j);
                }

                System.out.println("The out is:");
                for( int j : arr2 ){
                    System.out.println(j);
                }

                if (Arrays.equals(initOrder, arr2)){
                    System.out.println("The method is correct\n");
                }
                else {
                    System.out.println("The method isn't
correct\n");
                }

            }catch (IllegalArgumentException e){
                e.printStackTrace();
            }
        }


        /**
         * 以下代码为proof代码
         */

        for(int i =1;i<=8;++i){
```

```
            try {
                In fin3= new In("D:\\Study in SUSTech\\First
semester of junior year\\dsaaB\\lab\\assignment1\\assignment
1\\data\\"+i+".in");
                int[] destroyOrder= fin3.readAllInts();
                fin3.close();

                int[] initOrder = calculateInitCards(destroyOrder);
                for (int j : initOrder) {
                    System.out.print(j+" ");
                }
                System.out.print("\n");

                int[] destroy = calculateDestroyCards(initOrder);
                new File("D:\\Study in SUSTech\\First semester of
junior year\\dsaaB\\lab\\assignment1\\assignment
1\\data\\").mkdirs();
                try ( PrintWriter fout = new PrintWriter("D:\\Study
in SUSTech\\First semester of junior
year\\dsaaB\\lab\\assignment1\\assignment 1\\data\\"+i+".out") ) {
                    for( int j = 0; j < i*1000 && j <
destroy.length; ++ j )
                        fout.println(" " + destroy[j]);
                } catch ( IOException e ) {
                    e.printStackTrace();
                }

                if (Arrays.equals(destroyOrder,destroy)){
                    System.out.println("The method is correct");
                }
                else {
                    System.out.println("The method isn't correct");
                }


            }catch (IllegalArgumentException e){
                e.printStackTrace();
            }
        }
    }
}
```