บทที่ 10

ตัวอย่างเฉลยแบบฝึกหัดและข้อสอบ

ในบทนี้ก่อนอื่นผมจะเฉลยแบบฝึกหัดบางข้อจากบทเรียนเสียก่อน ในที่นี้จะเลือกเฉลยบทละหนึ่งข้อที่แสดงถึงเนื้อหาสำคัญของบท เนื่องจากต้องการให้นิสิตได้ฝึกฝนการทำด้วยตนเองโดยไม่พึ่งพาเฉลย (ถ้าทำไม่ออกจริงๆสามารถมาถามได้ตามปกตินะ) จากนั้นจะเป็นตัวอย่างข้อสอบที่ได้ใช้สอบจริงพร้อมเฉลยและคะแนนที่ให้ในแต่ละกรณี

เฉลยแบบฝึกหัด

บทที่ 1

ข้อ 20

ดูโค้ดแล้ว เราจะตอบค่า big O ได้ตามลำดับดังนี้ คือ $O(n^3)$, $O(n^2)$, $O(n^4)$, $O(n^4)$ โดยอันที่สองนั้นมาจากผลของอนุกรม ส่วนอันที่สามกับสี่นั้นต่างกันแค่ if statement ดังนั้นค่า big O จึงเท่ากัน

บทที่ 2

ข้อ 6

ในการเขียน insertion sort ขึ้นมาใหม่โดยใช้ recursion นั้น เนื่องจากโจทย์มีความกำกวม เราจึงต้องตั้งสมมติฐานเสียก่อน ก่อนอื่น ให้เป็นการจัดเรียงจำนวนเต็มบนอาร์เรย์ของจำนวนเต็มเท่านั้น

เราแยกโค้ดเป็นหลายๆส่วนตามขั้นตอนของ insertion sort ดังนี้ ก่อนอื่นมาคิดว่า insertion sort ต้องทำอย่างไร เราใช้หลักตามโค้ดข้างล่างนี้ สมมติว่าเรามีตัวแปรอาร์เรย์ซึ่งเห็นได้จากทุกเมธอด

```
public void insertionSort(int[] a){
    insertionSort(a, 0); //เรียกอันข้างล่าง
}

public void insertionSort(int[] a, int startIndex){
    //นี่คือ recursive insertion sort
    //startIndex คือตำแหน่งในอาร์เรย์ที่มีสมาชิกที่เราต้องการเอาไปยัดในที่ที่ถูกต้อง
    //โดยเราเริ่มดูจากตำแหน่งที่ 0 ก่อน จากนั้นใช้ recursive call ไล่ดูไปเรื่อยๆ

    if(startIndex >= a.length -1){
        return; //ถ้าตำแหน่งที่เราสนใจเลยอาร์เรย์แล้ว(เกิดจาก recursive call) ก็ไม่ต้องทำอะไร
    } else {
        //หาตำแหน่งที่ถูกต้องของค่า a[startIndex]
        int correctIndex = findAppropriatePlace(a, startIndex, a[startIndex]);

        //จากนั้นเลื่อนอาร์เรย์และยัดค่า a[startIndex] ลงในที่ที่ถูกต้องของมัน
        insert(a, startIndex, correctIndex);

        // แล้วก็เรียก insertion sort อีกที คราวนี้ดูที่ตำแหน่งถัดจาก startIndex
        insertionSort(a, startIndex+1);
}
```

ส่วนเมธอดย่อยต่างๆนั้นมีดังนี้

```
public int findAppropriatePlace(int[] a, int startIndex, int value){
    //startIndex -1 คือตำแหน่งที่เราจะเริ่มไล่หาย้อนอาร์เรย์ ว่าค่า value ควรจะไปอยู่ที่ไหน
    //เมธอดนี้รีเทิร์นตำแหน่งที่ value ควรจะถูกย้ายไป

    if(startIndex == 0){
        return 0; //ถ้าเราเริ่มดูจากตำแหน่งที่ 0 ก็แน่ละ ว่ายังไงก็ต้องยัด value ตรงนี้
    } else {
        if(value < a[startIndex-1]){
            //กรณีนี้ แสดงว่า value ยังย้ายต่อไปข้างซ้ายได้อีก จึงต้องตรวจสอบต่อ
            return findAppropriatePlace(a, startIndex-1, value);
        } else {
            return startIndex;
        }
    }
}

public void insert(int[] a, int elementIndex, int correctIndex){
    //elementIndex คือตำแหน่งในตอนแรก ส่วน correctIndex คือตำแหน่งจริงๆที่ควรจะอยู่

    int temp = a[elementIndex];
    shiftRight(a, elementIndex); //เลื่อนสิ่งที่อยู่ในอาร์เรย์ที่ต้องเลื่อนทั้งหมด
    a[correctIndex] = temp;  //จากนั้นจึงเอาค่าของเราใส่ในตำแหน่งที่ถูกต้อง
}

public void shiftRight(int[] a, int counter){
    if(counter == 0){
        return; // ไม่มีการเลื่อนอาร์เรย์แล้ว
    } else {
        a[counter] = a[counter-1];
```

```
        shiftRight(a, counter-1);
    }
}
```

บทที่ 3

ข้อ 4

สำหรับข้อนี้เป็นการทดสอบว่านิสิตมีพื้นฐานในการใช้อิเทอเรเตอร์เพียงใด ซึ่งจะต้องเรียกใช้เมธอดให้ถูกต้อง ถ้าทำข้อนี้ยังไม่ได้ก็ไม่สามารถแก้ปัญหาอื่นๆที่ใช้อิเทอเรเตอร์ได้ สำหรับโค้ดนั้นเป็นดังนี้

```
public LinkedListItr findPrevious(Object x){
    if(isEmpty()){
        return zeroth(); // ถ้าลิสต์ว่าง รีเทิร์นอิเทอเรเตอร์ที่ชี้โนดปลอม
    }

    //จะลงมาบรรทัดนี้ได้ อย่างน้อยลิสต์ต้องมีสมาชิกหนึ่งตัว
    LinkedListItr a = zeroth(); // สร้างอิเทอเรเตอร์ไว้ที่ดัมมี่โนด แล้วเริ่มไล่ดูจากจุดนี้
    while(a.current.next != null && !a.current.next.element.equals(x)){
        //a จะชี้ที่โนดท้ายลิสต์ไม่ได้ เพราะตำแหน่งถัดไปไม่มีแล้ว
        //และเวลาเทียบค่า ต้องเทียบของในโนดถัดจากที่ a ชี้

        a.advance();
    }
    return a;
}
```

```
public LinkedListItr find(Object x){
    if(isEmpty()){
        return new LinkedListItr(null); // ถ้าลิสต์ว่าง รีเทิร์นอิเทอเรเตอร์ที่ชี้ไปที่ null
    }
    //จะลงมาบรรทัดนี้ได้ อย่างน้อยลิสต์ต้องมีสมาชิกหนึ่งตัว
    LinkedListItr a = first(); // สร้างอิเทอเรเตอร์ไว้ที่โนดที่มีสมาชิกโนดแรก
    while(!a.isPastEnd() && !a.retrieve().equals(x)){
        a.advance();
    }
    if(a.current !=null){ //ถ้ามีให้ชี้ แสดงว่าเจอ x จริงๆ
        return a;
    } else { //ถ้าไม่มี x อยู่ในลิสต์เลย (ตัวอิเทอเรเตอร์ลูปจนตกลิสต์)
        return new LinkedListItr(null);
    }
}
```

บทที่ 4

ข้อ 3

ข้อนี้เป็นการทดสอบความรู้เรื่องการใช้งาน Map ซึ่งถ้าดูตามธรรมดาแล้ว จะต้องลูปโดยใช้อิเทอเรเตอร์ แต่ถ้านิสิตได้ศึกษาไลบรารี่ของ List Set Map มาเพียงพอก็จะรู้ว่าไม่จำเป็น โดยเราสามารถเปลี่ยนแม็ปให้เป็นเซต (เปลี่ยนได้ เนื่องจากแม็ปไม่มีของซ้ำอยู่แล้ว) แล้วทำงานบนเซตแทน ซึ่งโค้ดจะสั้นลงจากการใช้อิเทอเรเตอร์มาก โดยเป็นดังนี้

```
Set set1 = m1.entrySet();
Set set2 = m2.entrySet();
set2.removeAll(set1);
```

บทที่ 5

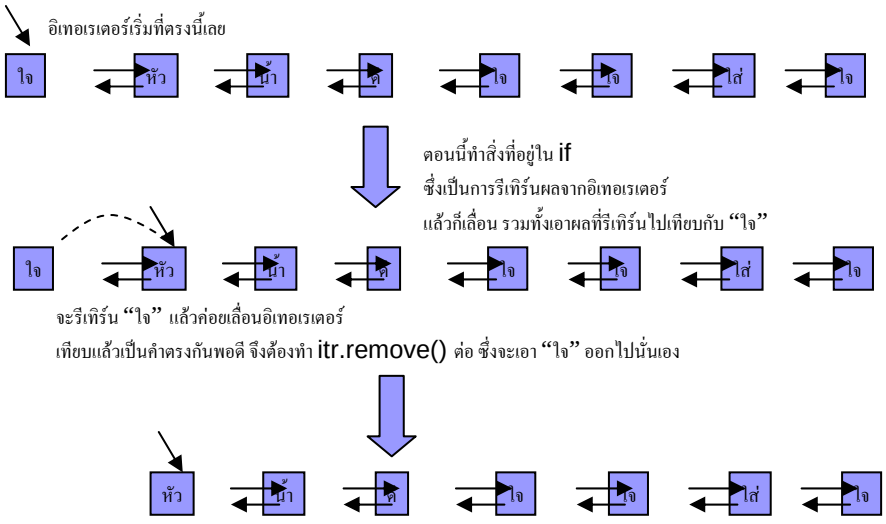
ข้อ 1

โค้ดนั้นจะเป็นดังนี้

```
ListIterator<String> itr = mylist.listIterator();
while(itr.hasNext()){
  if(itr.next().equals("ใจ"))
    itr.remove();
}
```

ซึ่งการทำงานนั้นแสดงในรูปข้างล่างนี้

สมมติตอนแรกนั้นลิสต์เป็นแบบนี้



ดังนั้นพอเราลูปไปเรื่อยๆ ก็จะสามารถเอา "ใจ" ออกจากลิงค์ลิสต์นี้ได้หมด

บทที่ 6

ข้อ 5 (ท้ายบท)

ข้อนี้จริงๆแล้วเป็นการประยุกต์ใช้คิวเอามารับข้อมูลจากโครงสร้างต้นไม้นั่นเอง ดังนั้นต้องเข้าใจทั้งโครงสร้างต้นไม้และคิว โดยหลักการคือ สร้างคิวของต้นไม้ระดับที่เราดูอยู่และระดับข้างล่างหนึ่งชั้น ไล่ดูคิวระดับที่เราอยู่แล้วสร้างระดับถัดไปเรื่อยๆ โดยโค้ดจะเป็นดังนี้

```
public LinkedList breadthFirst(BinaryNode t){
    if(t == null){
        return new LinkedList();
    }
    Queue thislevelQ = new Queue();
    thislevelQ.enqueue((Object)t);
    Queue nextlevelQ = new Queue();
    LinkedList result = new LinkedList();
    while(!thislevelQ.isEmpty()){
        //เอาสิ่งแรกใน thislevelQ ออกมา
        BinaryNode thenode = (BinaryNode) thislevelQ.dequeue();

        result.insert((Object)elementAt(thenode), last()); // last() นี่เป็นเมธอดของลิสต์นะ
        if(thenode.left !=null)
            nextlevelQ.enqueue((Object)thenode.left);
        if(thenode.right !=null)
            nextlevelQ.enqueue((Object)thenode.right);
        if((thislevelQ.isEmpty()) && (!nextlevelQ.isEmpty())){
            thislevelQ = nextlevelQ;
            nextlevelQ = new Queue();
        }
    }
```

```
    return result;
}
```

บทที่ 7

ข้อ 2

```
private AvLNode findKth(int k, AvlNode t){
    if(t == null)
        return null;
    if(k>(t.size+1)) //ตัวที่ k เลย tree ไป
        return null;
    if(k==(t.size+1)) //k คือตำแหน่งสุดท้าย
        return findMax(t);
    if(k ==1) // k คือตัวแรกของ tree
        return findMin(t);

    //คราวนี้ดูว่า left ต้องไม่ empty จึงจะ search ใน left ได้
    int nodeInLeftSubtree = 0; // default ไว้ เผื่อเจอ empty subtree
    if(t.left!=null)
        nodeInLeftSubtree = t.left.size+1;
    if(nodeInLeftSubtree>=k) // อยู่ใน left แน่ๆ
        return findKth(k,t.left);
    if(nodeInLeftSubtree+1 == k) //ตัว root คือตัวของ k
        return t;
    if( nodeInLeftSubtree+1<k) //k อยู่ใน right แน่ๆ
        returnfindKth(k-(nodeInLeftSubtree+1), t.right);
}
```

บทที่ 8

ข้อ 3

โค้ดนั้นไม่ยากอย่างที่คิด เพราะหลักการของฮีปก็คือ ถ้าเราไม่มั่นใจ ก็เอาของออกจากฮีปเก่าทีละตัวมาสร้างฮีปใหม่เท่านั้นเอง ดังนั้นโค้ดจึงเป็นดังนี้

```
Heap mergeHeap(Heap theSecondHelp){
    while(theSecondHeap.size>0){
        This.add(theSecondHeap.removeMin());
    }
    return this;
}
```

เรื่อง big O อาจจะมากสักหน่อย เพราะเราประกอบฮีปใหม่ทั้งหมด โดยถ้าให้ this มีสมาชิก n ตัว ส่วน theSecondHeap มีสมาชิก m ตัว ค่า big O รวมจะ

= (big O ของการ removeMin จาก theSecondHeap + big O ของการ add ไปที่ this) * m

= (O(log m) + O(log(n+m))) * m = m(log (n+m))

บทที่ 9

ข้อ 1

ผลในตารางแฮชจะเป็นดังนี้

		19	20		37			36	53					54		

โดย 20 จะเอาใส่ได้เลย

37 จะชนหนึ่งครั้ง กับ 20

54 ชนหนึ่งครั้งกับ 20

19 เอาใส่ได้เลย

36 ชนสองครั้ง ครั้งแรกกับ19 ครั้งที่สองกับ37

53 ชนสองครั้ง ครั้งแรกกับ 19 ครั้งที่สองกับ 54

ตัวอย่างข้อสอบพร้อมเฉลย

สำหรับตัวอย่างข้อสอบนั้น จะมีตัวข้อสอบ เฉลย และคะแนนที่ให้แต่ละส่วนของแต่ละข้อ

1. (7 คะแนน) จงเขียนโค้ดของการทำ selection sort ของอาร์เรย์ของจำนวนเต็ม (7 นาที)

```
selectionSort(int[] a){
    int maxIndex = 0 ;
    int unsorted = a.length; //นี่คือจำนวนสมาชิกที่ต้องตรวจ
    while(unsorted > 0){
        maxIndex = 0;
        for(int i = 0; i < unsorted; i++){
            if(a[i] > a[maxIndex]){
                maxIndex = i;
            }
        }// ได้ maxIndex ตรงนี้
        if(a[maxIndex]!=a[unsorted-1]){ // ถ้าตัว maxIndex ไม่อยู่ที่ท้ายสุดที่ต้องดู (ถ้าอยู่จะไม่ต้องสลับที่)
            swap(a, maxIndex, unsorted-1);
        }
        unsorted- -;
    }
}
```

Initialize 1 ถ้าไม่ครบหักหมด

Condition ถูกต้องอันละ 1

มี while 0.5 ที่ลูปได้จริง

1

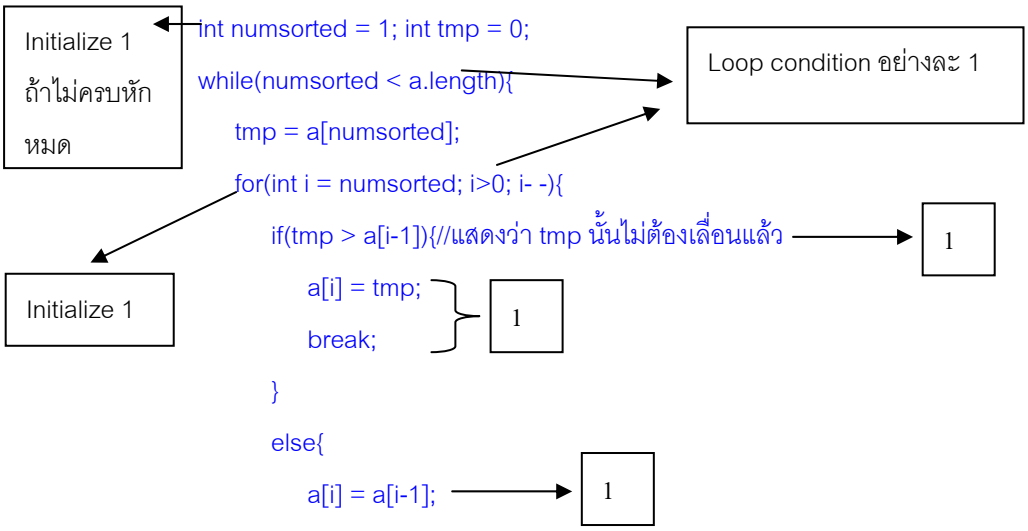
1

มี for 0.5

1

2. (7 คะแนน) จงเขียนโค้ดของการทำ insertion sort ของอาร์เรย์ของจำนวนเต็ม (13 นาที)

```
insertionSort(int[] a){
    int numsorted = 1; int tmp = 0;
    while(numsorted < a.length){
        tmp = a[numsorted];
        for(int i = numsorted; i>0; i- -){
            if(tmp > a[i-1]){//แสดงว่า tmp นั้นไม่ต้องเลื่อนแล้ว
                a[i] = tmp;
                break;
            }
            else{
                a[i] = a[i-1];
            }
        }
        numsorted++;
    }
}
```



3. (7 คะแนน) ถ้าต้องการเปลี่ยนโค้ดของ insertion sort ในข้อ 2 ให้ทำการเรียงของในอาร์เรย์ โดยให้จำนวนคี่มาก่อนจำนวนคู่ นั่นคือ

- ถ้ามีจำนวนคี่สองตัว ตัวที่มีค่าน้อยจะเรียงอยู่ก่อน
- ถ้ามีจำนวนคู่สองตัว ตัวที่มีค่าน้อยจะเรียงอยู่ก่อน
- ถ้ามีจำนวนหนึ่งเป็นจำนวนคี่ อีกจำนวนหนึ่งเป็นจำนวนคู่ จำนวนคี่จะถูกเรียงอยู่ก่อน

ถามว่า จะต้องเปลี่ยนโค้ดในข้อ 2 เป็นอย่างไร (7 นาที)

วิธีทำคือแค่เขียนนิยามของ “น้อยกว่า” ขึ้นมาแล้วเอามาใช้แทนเครื่องหมาย < ในโค้ดหน้าที่แล้ว

```
boolean lessThan(){int[] a, int i, int j){
    int num1, num2;
    num1 = a[i];
    num2 = a[j];
    if(num1%2 == num2%2){// เป็นคู่ทั้งคู่หรือคี่ทั้งคู่
        return(num1 < num2);
    }
    else if(num1%2 == 1 && num2%2 == 0){
          //num1 เป็นคี่ num2 เป็นคู่ คี่นั้นน้อยกว่า ตอบ true เลย
        return true;
    }
    else{
        return false;
    }
}
```

ข้อนี้เขียนยังไงก็ได้ให้ถูก ถ้าผิดหัก ที่ละ 1

4. มีสแตกของจำนวนเต็มอยู่ ซึ่งมีเมธอดดังนี้

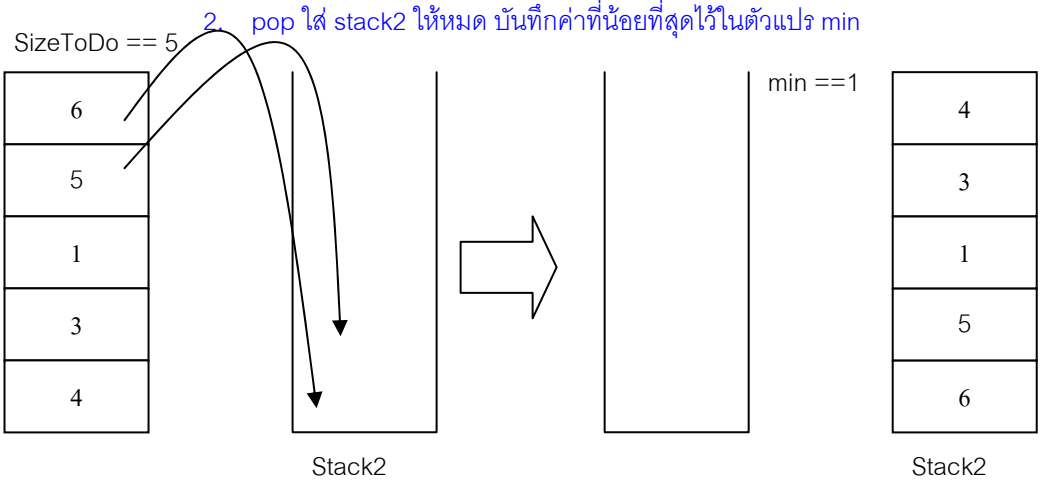
- public void push(int x); // เอา x ใส่ด้านบนสุด
- public int pop(); // เอา ของที่อยู่บนสุดของสแตกออกมา รีเทิร์นค่าของมันด้วย
- public int size(); // รีเทิร์นจำนวนสมาชิกปัจจุบันในสแตก

ถ้าเราจะจัดเรียงของในสแตกที่เก็บจำนวนเต็มให้เรียงจากมากไปน้อย (ให้ค่ามากอยู่ด้านบนของสแตก) โดยสร้างตัวแปรได้ สร้างสแตกเพิ่มเติมได้ แต่ห้ามใช้อาร์เรย์ และให้ใช้เมธอดของสแตกเท่านั้น

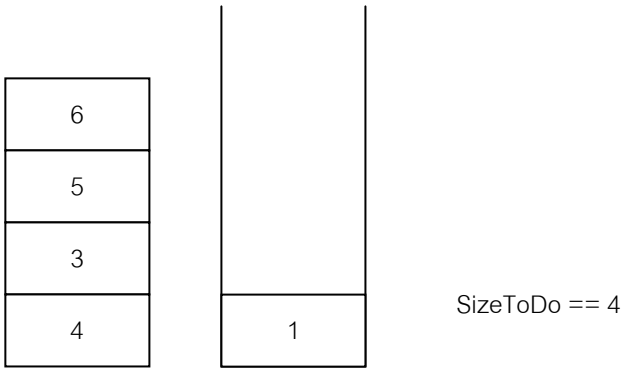
ยังไงก็ได้ ถ้าเวิร์คจริงในกรณีทั่วไป รวมทั้ง base case ก็ให้ 4

- จงอธิบายวิธีการแก้ปัญหานี้ พร้อมวาดรูปประกอบ (4 คะแนน)
- จงเขียนเมธอดเพื่อทำตามวิธีที่เสนอนี้ (5 คะแนน) (เวลารวมทั้งข้อ 4 นี้คือ 33 นาที)

1. ก่อนอื่นสร้างตัวแปร sizeToDo = size() เก็บไว้ แล้วสร้าง stack2 ขึ้นมาอีกตัว
2. pop ใส่ stack2 ให้หมด บันทึกค่าที่น้อยที่สุดไว้ในตัวแปร min



3. pop ของกลับไปที่สแตกตัวแรกเป็นจำนวน sizeToDo ชิ้น โดยลบตัวที่เป็น min ทิ้งไปเลย (แต่ค่าของมันยังเก็บไว้ในตัวแปร min)
   i. หลังจากนั้นเอา min push ใส่ stack2
   ii. และลดค่าของ sizeToDo ลงมาหนึ่ง

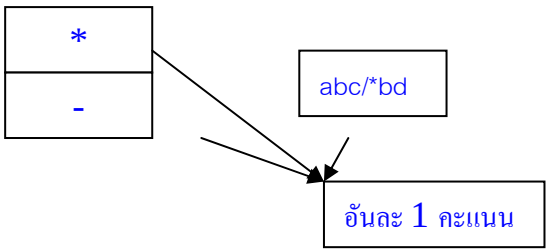


4. กลับไปทำข้อ 1 ใหม่ วนทำเรื่อยๆ (ตัวแปรควบคุมคือ sizeToDo)

```
public void sortStack(){// ให้นี่เป็นเมธอดของสแตกเอง
    Stack s2 = new Stack();
    int sizeToDo = size();
    int min = 999; // sentinel
    while(sizeToDo > 0){
        for(int i = 1; i <= sizeToDo; i++){
            int tmp = pop();
            s2.push(tmp);
            if(tmp < min){
                min = tmp;
            }
        }// loop to push things from s1 to s2 and find min at the same time
        for(int j=1; j <= sizeToDo; j++){
            int tmp = s2.pop(); //
            if(tmp != min){
                push(tmp);
            }
        }// pop back to first stack, except min
        s2.push(min); //push min into stack2
        sizeToDo- -;
      }
}
```

อันละ 0.5

5. (3 คะแนน) ถ้าเรามีเลขให้คิดดังนี้ คือ a*(b/c) – b*d ถามว่า ถ้าเราใช้สแตกแปลงเลขนี้ให้เป็น postfix เมื่ออ่านอินพุตนี้หมดพอดี จะมีเอ้าท์พุตเป็นอะไร และเหลืออะไรอยู่บนสแตก (1 นาทีเอง)



6. (6 คะแนน) เรามีคิวที่มีเมธอดดังนี้

void enqueue(Object x); //เอา x ต่อคิว

Object dequeue(); //เอาของออกจากคิว รีเทิร์นของนั้นออกมา

int size(); // รีเทิร์นจำนวนสมาชิกในคิว

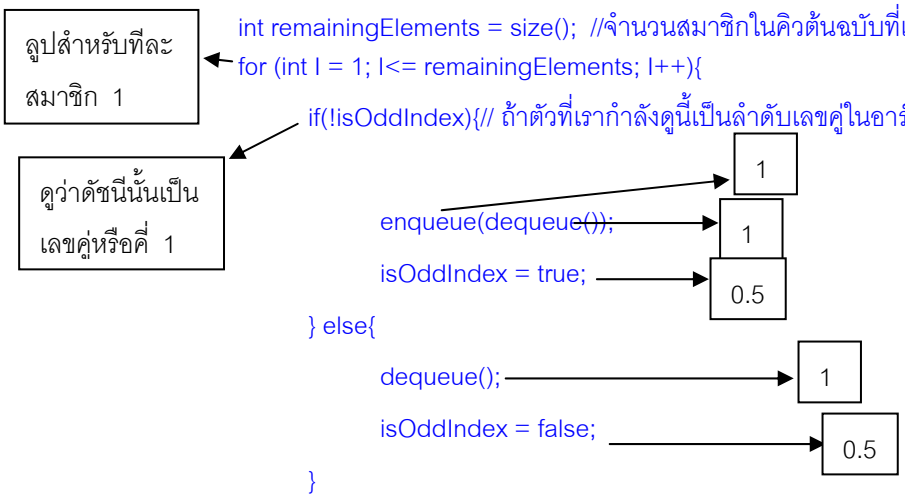
ให้ถือว่า สมาชิกที่อยู่ที่หัวคิวมี index == 0 จงเขียนเมธอดของคิวเพิ่ม คือ

void removeOddIndex(); //เอาสมาชิกที่มีลำดับ index เป็นเลขคี่ ทิ้งออกจากคิวไปให้หมด

นั่นคือ ถ้าคิวมีสมาชิกเป็น a,b,c,d,e,f เมธอดนี้จะทำให้คิวกลายเป็น a,c,e

ในการเปลี่ยนแปลงคิว ให้ใช้เมธอดของคิวเท่านั้น และห้ามสร้างตัวแปรอื่นนอกจากตัวแปรที่เป็นชนิดพื้นฐาน (primitive type) (5 นาที)

```
void removeOddIndex(){
        boolean isOddIndex = false; //บอกว่าสมาชิกของคิวเดิมมีindex เลขคี่ในคิว
ต้นฉบับหรือไม่
        int remainingElements = size();  //จำนวนสมาชิกในคิวต้นฉบับที่เรายังไม่ได้ดู
        for (int I = 1; I<= remainingElements; I++){
                if(!isOddIndex){// ถ้าตัวที่เรากำลังดูนี้เป็นลำดับเลขคู่ในอาร์เรย์ต้นฉบับ
                        enqueue(dequeue());
                        isOddIndex = true;
                } else{
                        dequeue();
                        isOddIndex = false;
                }
        }
}
```



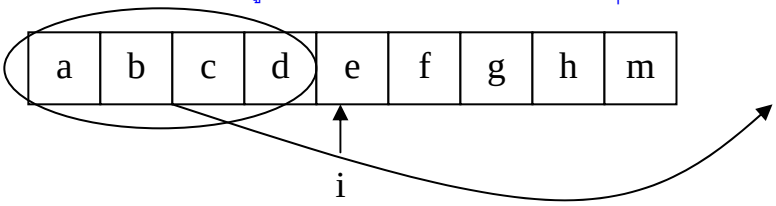
7. (11 คะแนน) ถ้าเรามีคิวที่มีเมธอดเหมือนดังข้อ 6 จงเขียนโค้ดของเมธอดของคิวต่อไปนี้

public void jumpQueue(Object x, int index)

เมธอดนี้เอา x แทรกที่ตำแหน่ง index โดยสมาชิกตัวเก่าที่ตำแหน่ง index นี้ต้องเลื่อนออกไปอยู่หลัง x

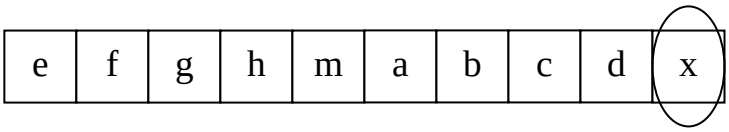
จงอธิบายสิ่งที่โค้ดนี้ทำด้วยพร้อมวาดรูปประกอบ ไม่อนุญาตให้สร้างคิวเพิ่ม (13 นาที)

1. ถ้า index ยังไม่ถึง i ให้ลูป dequeue กับ enqueue ไปเรื่อยๆ

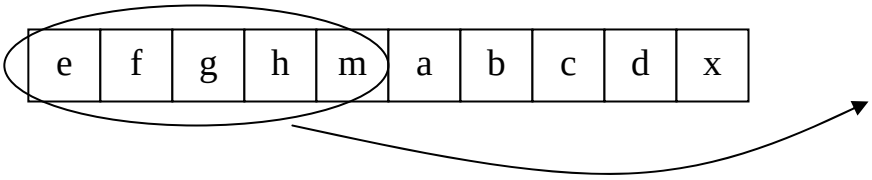


อธิบายครบถ้วน สมบูรณ์เอาไป 4

2. พอถึงตรงที่นับ index ถึง i ให้ยัด x ลงไป ในคิวด้วยการ enqueue



3. แล้วเอาที่เหลือ ทำ ลูป dequeue กับ enqueue ไปเรื่อยๆจนครบ แต่ต้องนับเพิ่มอีก 1 ครั้ง เพราะเราไม่ได้ dequeue ตอนที่ใส่ x เข้าไป ซึ่งทำให้จำนวนครั้งผิดไป




```
Int size = size();
If(index > size || index < 0 ){
    Throw new Exception();
}
// from now, we are sure that the given index is legal
for(int i=0; i<= size; i++){
    if (i!= index){// ถ้าเรากำลังดูตรงที่ไม่ต้องแทรก
        enqueue(dequeue());
    } else { // i == index
        enqueue(x);
    }
}
```

1 1 1

เพิ่มการนับไปหนึ่งครั้ง บรรทัดนี้ 1 คะแนน

1 1 1

8. (10 คะแนน) ถ้าเรามีลิงค์ลิสต์ตามรูปแบบของ Mark Allen Weiss (ลิงค์ลิสต์ประกอบด้วยคลาส ListNode, LinkedListItr และ LinkedList ดังรูปที่ 1 (บางเมธอดจะมีแค่คำอธิบายเท่านั้น แต่ให้เรียกใช้ได้ทุกเมธอด) และให้ออบเจ็กต์เปลี่ยนเป็นจำนวนเต็มได้ด้วยการเรียกเมธอด intValue()) สมมุติว่าเรามีลิงค์ลิสต์อยู่สองลิสต์ดังนี้

- C เป็นลิงค์ลิสต์ใดๆ
- P เป็นลิงค์ลิสต์ที่มีสมาชิกเป็นจำนวนเต็มที่จัดเรียงจากน้อยไปมาก

จงเขียนเมธอด LinkedList specificElements(C, P) ซึ่งจะสร้างลิงค์ลิสต์ใหม่จาก C โดยดึงสมาชิกมาตามที่บอกไว้ใน P นั่นคือ ถ้า P มีสมาชิกคือ 1,3,4,6 ลิงค์ลิสต์ตัวใหม่จะเกิดจากการเอาสมาชิกตัวที่ 1,3,4 และ 6 ของ C มาสร้าง (42 นาที)

```
LinkedList specificElements(C, P){
    LinkedList new = new LinkedList();
    LinkedListItr itr1 = c.first();
    LinkedListItr itr2 = p.first();
    LinkedListItr itr3 = new.zeroth();
    int currentIndex1 =0; // index ปัจจุบันของลิสต์แรกที่เรากำลังสนใจอยู่
    while(!itr2.isPastEnd()){
        Integer x = (Integer)itr2.retrieve();
        int x2 = x.intValue();
        while(!itr1.isPastEnd() && currentIndex1 < x2){
            itr1.advance();
            currentIndex1++;
        }
        if(itr1.isPastEnd()){ // ถ้าออกมาโดยสืบหาเลยลิสต์แรกไปแล้ว
            // แสดงว่าไม่ต้องทำต่อแล้ว รีเทิร์นลิสต์คำตอบได้เลย
            return new;
        } else{ // ถ้าชี้ไปยังตัว index นั้น (มีตัวให้เอาออกมาใส่ลิสต์ผลลัพธ์)
```

1 ถ้าไม่ครบ ก็ 0

1 condition ต้องถูกหมด

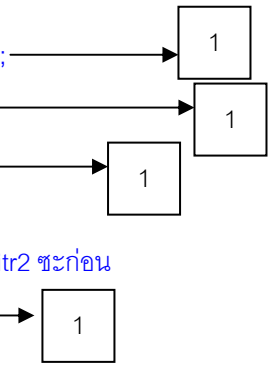
1 ต้องมีการเอาตัวเลข index จาก p มาใช้

1 condition ต้องถูกหมด

1 ต้องครบ

1

```
            // ก็เอาออกมาใส่ลิสต์ผลลัพธ์ซะ
            Object     element = itr1.retrieve();
            new.insert(element, itr3);
            itr3.advance();
        }
        // ก่อนจะทำลูปนอกสุดต่อไป ก็ต้องเพิ่ม itr2 ซะก่อน
        itr2.advance();
          }// end while
          // ถ้ามาถึงตรงนี้ได้ แสดงว่าไม่มี index ให้เอามาอีกแล้ว รีเทิร์นลิสต์คำตอบไปเลย
    return new;
}
```



```
1:  class ListNode{
2:          Object   element;
3:          ListNode next;
4:          // Constructors
5:          ListNode( Object theElement )
6:          {
7:              this( theElement, null );
8:          }
9:
10:         ListNode( Object theElement, ListNode n )
11:         {
12:             element = theElement;
13:             next    = n;
14:         }
15: }
16:
17:    public class LinkedListItr{
18:         ListNode current;    // ตำแหน่งปัจจุบันที่เราสนใจ
19:         LinkedListItr( ListNode theNode )
20:         {
21:             current = theNode;
22:         }
23:
24:         /**
25:          * ดูว่า current เลยท้ายลิสต์ไปหรือยัง
26:          * @return true ถ้า current เป็น null
27:          */
28:         public boolean isPastEnd( );
29:
30:         /**
31:          * @return item ที่เก็บไว้ใน current หรือไม่ก็ null ถ้าตำแหน่งของ
32:          * current ไม่ได้อยู่ในลิสต์
33:          */
34:         public Object retrieve( );
35:
36:         /**
37:          * เขยิบ current ไปยังตำแหน่งถัดไปในลิสต์ ถ้า current เป็น null ก็ไม่ต้อง
38:                                          *    ทำอะไร
39:          */
40:         public void advance( );
41:    }
42:
43:   public class LinkedList{
44:     ListNode header; //เข้าได้จากหัวลิสต์
45:     public LinkedList( ){
46:             header = new ListNode( null );
47:     }
48:
49:     public boolean isEmpty( ){
50:         return header.next == null;
51:     }
52:
53:     public void makeEmpty( ){
54:         header.next = null;
55:     }
56:
57:     /**
58:      * รีเทิร์น iterator ที่ชี้ไป header node.
```

```
59:      */
60:     public LinkedListItr zeroth( );
61:
62:
63:     /**
64:      * รีเทิร์น iterator ที่ชี้ไป node ถัดจาก header (ซึ่งเป็น null ได้
65:                      * ถ้าลิสต์นี้ว่าง)
66:      */
67:     public LinkedListItr first( );
68:
69:     /**
70:      * ใส่โนดใหม่ตามหลังสมาชิกที่ชี้ด้วย p
71:      * @param x item ที่จะเอาใส่โนดใหม่
72:      * @param p เป็น iterator ชี้ตำแหน่งที่อยู่ก่อนโนดที่จะลงใหม่
73:      */
74:     public void insert(Object x,LinkedListItr p);
75:
76:     /**
77:      * @param x คือ ของข้างในของโนดที่เราต้องการหา
78:      * @return iterator ที่ชี้ไปที่โนดแรกที่มี x อยู่ข้างใน หรือชี้ไปที่ null ถ้า x
79:                       *ไม่อยู่ ในลิสต์เลย
80:     /*
81:     public LinkedListItr find(Object x);
82:
83:     /**
84:     * รีเทิร์น iterator ที่ชี้ไปที่โนดก่อนโนดแรกที่มี x
85:     * ถ้าไม่มี x ในลิสต์เลยให้รีเทิร์น iterator ที่ชี้ไปที่โนดสุดท้ายของลิสต์
86:     */
87:     public LinkedListItr findPrevious(Object x);
88:
89:     /**
90:      * เอาโนดของ x ตัวแรกที่เจอออกจากลิสต์
91:      * @param x คือ item ในโนดที่ต้องการเอาออก
92:      */
93:      public void remove(Object x);
94:
95:      /**
96:      * เอาโนดที่อยู่ถัดจาก header node ทิ้งไป แต่ส่วนอื่นของลิสต์คงเดิม
97:      * @return ออบเจ็กต์ที่อยู่ในโนดที่เราทิ้งไป
98:      */
99:      public Object removeFirst();
100: }
```

รูปที่ 1 ListNode และคลาสอื่นๆของลิงค์ลิสต์พื้นฐาน

9. (6 คะแนน) สมมติว่าเรามีเซตอยู่สามเซตคือ J B และ D จงเขียนเมธอดที่รีเทิร์นค่า (J intersect B) – D โดยใช้ Java Collection Framework (เมธอดของ Set และ List จาก Java Collection Framework นั้นอยู่ในรูปที่ 2) เซตต้นฉบับจะต้องไม่เปลี่ยนแปลง (3 นาที)

```
Set s1 = new HashSet(J);
Set s2 = new HashSet(B);
Set s3 = new HashSet(D);
s1.retainAll(s2);
s1.removeAll(s3);
return s1;
```

บรรทัดละ 1 คะแนน

10. สมมติว่าเราจะเอาลิงค์ลิสต์ใน Java Collection Framework ที่มีข้อมูลชนิด MyType ไปทำการทำ quicksort

    a. (8 คะแนน) จงเขียนโค้ดของเมธอดของลิงค์ลิสต์ดังนี้ (12 นาที)

    public ListIterator<MyType> findPivot();

    ซึ่งจะหา pivot จากหลักการ median of three (ให้ throw exception ถ้าลิสต์ว่าง) โดยรีเทิร์นลิสต์อิเทอเรเตอร์ของตำแหน่ง pivot นั้น

```
public ListIterator<MyType> findPivot() throws Exception{
    int size = size();
    if(isEmpty()){  // ถ้าลิสต์ว่างเราจะไม่สามารถ sort ได้
        throw Exception();
    }
        // มาถึงบรรทัดนี้ ไม่ empty แน่ๆ
    MyType a = get(0);
    MyType b = get(size-1);
    MyType c = get((size-1)/2);
    If ( (a.compareTo(b) < 0 && c.compareTo(a) < 0) ||  (a.compareTo(c) < 0 &&
b.compareTo(a) < 0 ) ){
```

1

1 ต้องครบ ไม่งั้น 0

1.5 , ทั้งบรรทัดต้องครบไม่งั้น 0

```
        ListIterator<MyType> itr = listIterator();
        return itr;
          }
    else if((a.compareTo(c) < 0 && c.compareTo(b) < 0) ||  (b.compareTo(c) < 0 &&
c.compareTo(a) < 0 )){
        ListIterator<MyType> itr = listIterator((size-1)/2);
        return itr;
    } else {
        ListIterator<MyType> itr = listIterator(size-1);
        return itr;
    }
}
```

1

1.5 , ทั้งบรรทัดต้องครบไม่งั้น 0

1

1

b. (4 คะแนน) จงเขียนโค้ดของเมธอดของลิงค์ลิสต์ดังนี้

public void swap(ListIterator<MyType> i, ListIterator<MyType> j);

เมธอดนี้สลับที่สมาชิกสองตัวในลิสต์ โดยแต่ละตัวนั้นถูกชี้ด้วยอิเทอเรเตอร์ (4 นาที)

```
public void swap(ListIterator<MyType> i, ListIterator<MyType> j){
        MyType a = i.next(); // เก็บสมาชิกที่ตำแหน่ง i
        MyType b = j.next();
        i.set(b);
        j.set(a);
}
```

Code บรรทัดละคะแนน

c. (7 คะแนน) จงเขียนโค้ดของเมธอดของลิงค์ลิสต์ดังนี้ (14 นาที)

public void partition(ListIterator itr);

รับอิเทอเรเตอร์ที่บอกตำแหน่งของ pivot ต่อจากนั้นทำการแบ่ง partition โดยให้สมาชิกที่น้อยกว่า pivot ไปอยู่ด้านซ้ายของ pivot และ สมาชิกที่มากกว่า pivot ไปอยู่ด้านขวาของ pivot

```
public void partition(ListIterator itr){
        int i=0;
        int size = size();
        int j = size-2;
        ListIterator<MyType> l = listIterator(size-1);
        swap(itr,l);
        while(true){
            while((i < size-1) && get(i).compareTo(get(size-1)) <0){
                i++;
            }
            // ต่อไปเลื่อน j บ้าง
            while(j>=0 && get(j).compareTo(get(size-1)) >0){
                j- -;
            }
            if(j<i){ // เกยกันแล้ว ต้องสลับที่ i กับ pivot
                ListIterator<MyType> li = listIterator(i);
                swap(li,l);
                return;
            } else {
                ListIterator<MyType> li = listIterator(i);
                ListIterator<MyType> lj = ListIterator(j);
                swap(li,lj);
                j - - ;
                i++;
            }
        }
}
```

pivot

1

1

1 ต้องครบ ไม่งั้น 0

1 ต้องครบ ไม่งั้น 0

2 ต้องครบ ไม่งั้น 0

1 ต้องครบ ไม่งั้น 0

Set Interface

boolean add(E o) เติมสมาชิกใหม่ลงไปในเซตถ้ามันไม่ซ้ำกับสมาชิกที่มีอยู่แล้ว นั่นคือ เติม o ลงไปในเซตถ้าไม่มี e ซึ่ง (o==null ? e==null : o.equals(e)) ถ้ามีสมาชิกตัวที่เราต้องการเติมอยู่ในเซตแล้ว เซตจะไม่เปลี่ยนแปลงและเมธอดนี้จะรีเทิร์น false
boolean addAll(Collection<? extends E> c) เติมสมาชิกจากคอลเลคชั่น c ทั้งหมดลงในเซตของเรา แต่เมธอดนี้จะถือว่าใช้ไม่ได้ถ้า c ถูกเปลี่ยนแปลงระหว่างที่เมธอดนี้ทำงานอยู่ นั่นก็คือ c จะเป็นเซตที่เราใช้เรียกเมธอดนี้เองไม่ได้ เมธอดนี้รีเทิร์น true ถ้าการเรียกเมธอดนี้ทำให้ภายในเซตเปลี่ยนไป พารามิเตอร์ <? extends E> หมายถึงอะไรก็ได้ที่เป็นสับคลาสของ E โดย E คือพารามิเตอร์ของเซตที่เรียกใช้เมธอดนี้ อย่าลืมว่าคลาสใดๆก็เป็นสับคลาสของตัวมันเอง อย่างหนึ่งที่ต้องรู้คือ เมธอดนี้ไม่เติมสมาชิกที่ซ้ำ นั่นคือ ถ้า c เป็นเซต เราก็จะได้การยูเนี่ยนของสองเซตนั่นเอง
void clear() เอาสมาชิกของเซตนี้ออกไปให้หมด
boolean contains(Object o) รีเทิร์น true ถ้าเซตนี้มี o อยู่ หรือพูดได้อีกอย่างว่า รีเทิร์น true ก็ต่อเมื่อเซตนี้มีสมาชิก e ซึ่ง (o==null ? e==null : o.equals(e)).
boolean containsAll(Collection<?> c) รีเทิร์น true ถ้าเซตนี้มี สมาชิกจากคอลเลคชั่น c อยู่ทั้งหมด
boolean equals(Object o) รีเทิร์น true ถ้า o เป็นเซตที่มีขนาดเดียวกับเซตที่เรียกใช้เมธอดนี้และทุกสมาชิกที่อยู่ใน o อยู่ในเซตที่เรียกใช้เมธอดนี้ อย่าลืมว่า o ต้องเป็น Set ด้วย
int hashCode() รีเทิร์นค่าแฮชโค้ด ซึ่งแฮชโค้ดของเซต คือผลบวกของแฮชโค้ดจากสมาชิกในเซต (แฮชโค้ดของ null คือ 0)
boolean isEmpty()

รีเทิร์น true ถ้าเซตนี้ไม่มีสมาชิก
Iterator<E> iterator() รีเทิร์นอิเทอเรเตอร์ การเรียงของสมาชิกจะขึ้นอยู่กับว่าจริงๆแล้วเซตนี้เป็นแฮชเซตหรือทรีเซต
boolean remove(Object o) เอาสมาชิก e ซึ่ง (o==null ? e==null : o.equals(e)) ออกจากเซต ถ้ามีอยู่ในเซต รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ
boolean removeAll(Collection<?> c) เอาสมาชิกที่เหมือนกับสมาชิกใน c (โดยเทียบด้วย equals()) ทิ้ง รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ c ห้ามเป็น null ไม่งั้นจะ throw exception
boolean retainAll(Collection<?> c) เอาสมาชิกที่เหมือนกับสมาชิกใน c (โดยเทียบด้วย equals()) เหลือไว้ นอกนั้นลบทิ้งหมด รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ c ห้ามเป็น null ไม่งั้นจะ throw exception
int size() รีเทิร์นจำนวนสมาชิกในเซตนี้ ถ้าจำนวนสมาชิกมากกว่า Integer.MAX_VALUE ให้รีเทิร์น Integer.MAX_VALUE.
Object[] toArray() รีเทิร์นอาร์เรย์ซึ่งใส่สมาชิกของเซตนี้ไว้ทั้งหมด ถ้ามีการกำหนดลำดับของสมาชิกด้วยอิเทอเรเตอร์ไว้แล้ว ลำดับสมาชิกในอาร์เรย์ก็ต้องเป็นไปตามนั้นด้วย
<T> T[] toArray(T[] a) รีเทิร์นอาร์เรย์ซึ่งใส่สมาชิกของเซตนี้ไว้ทั้งหมด รันไทม์ไทป์ของอาร์เรย์ที่รีเทิร์นให้เป็นชนิดเดียวกับอาร์เรย์ a ถ้าเซตของเราใส่ a ได้ ก็จะเอาใส่ a แล้วรีเทิร์น a เลย (สมาชิกในด้านหลังของ a ที่มีที่เหลือก็จะถูกเซ็ตเป็น null) มิฉะนั้นต้องสร้างอาร์เรย์ใหม่ซึ่งขนาดใหญ่พอที่จะเก็บเซตของเราได้ ถ้ามีการกำหนดลำดับของสมาชิกด้วยอิเทอเรเตอร์ไว้แล้ว ลำดับสมาชิกในอาร์เรย์ก็ต้องเป็นไปตามนั้นด้วย

List Interface

boolean add(E o) ใส่ o ไปที่ท้ายลิสต์ ลิสต์ที่เราสร้างขึ้นอาจกำหนดชนิดของข้อมูลที่ใส่ได้เอาไว้ เช่น ไม่รับค่า null หรือไม่รับข้อมูลบางชนิด เอกสารของลิสต์แต่ละชนิดควรบอกถึงชนิดของข้อมูลที่ไม่รับด้วย รีเทิร์น true ถ้าการเรียกเมธอดนี้ทำให้ภายในคอลเลคชั่นเปลี่ยนไป
void add(int index, E element) ใส่ element เข้าไปที่ตำแหน่งที่ index เลื่อนสมาชิกในลิสต์ตัวที่เคยอยู่ตรงนั้น รวมทั้งสมาชิกตัวถัดไปอื่นๆ ไปทางขวาตัวละตำแหน่ง
boolean addAll(Collection<? extends E> c) ใส่ของใน c ต่อท้ายลิสต์ ลำดับที่ใส่นั้นเป็นไปตามอิเทอเรเตอร์ของ c เมธอดนี้จะใช้ไม่ได้ถ้า c ถูกเปลี่ยนแปลงในระหว่างที่เรียกใช้เมธอดนี้ เช่นเมื่อ c คือตัวลิสต์เอง รีเทิร์น true ถ้ามีการเติมสมาชิกลงไปจริง
boolean addAll(int index, Collection<? extends E> c) ใส่ของใน c ลงไปในลิสต์ ณ ตำแหน่งที่ถูกกำหนดโดย ลำดับที่ใส่นั้นเป็นไปตามอิเทอเรเตอร์ของ c เลื่อนสมาชิกในลิสต์ตัวที่เคยอยู่ตำแหน่งนั้น รวมทั้งสมาชิกตัวถัดไปอื่นๆ ไปทางขวา เมธอดนี้จะใช้ไม่ได้ถ้า c ถูกเปลี่ยนแปลงในระหว่างที่เรียกใช้
void clear() ทำให้ลิสต์นี้ว่าง
boolean contains(Object o) รีเทิร์น true ถ้าลิสต์นี้มี o อยู่ หรือพูดได้อีกอย่างว่า รีเทิร์น true ก็ต่อเมื่อคอลเลคชั่นนี้มีสมาชิก e ซึ่ง (o==null ? e==null : o.equals(e))
boolean containsAll(Collection<?> c) รีเทิร์น true ถ้าลิสต์นี้มี สมาชิกจากคอลเลคชั่น c อยู่ทั้งหมด
boolean equals(Object o) เปรียบเทียบออบเจ็กต์ o กับลิสต์นี้ว่าเท่ากันหรือไม่ รีเทิร์น true ถ้า o ก็เป็นลิสต์ ลิสต์ของเรากับ o มีขนาดเท่ากัน และมีสมาชิกเหมือนกัน (เทียบด้วยเมธอด equals())

เรียงกันด้วยลำดับเดียวกันทุกประการ
E get(int index) รีเทิร์นสมาชิก ณ ตำแหน่งที่บอกด้วย index

int hashCode()
รีเทิร์นแฮชโค้ดของลิสต์นี้ โดยมีสูตรว่า
hashCode = 1;
Iterator i = list.iterator();
while (i.hasNext()){
Object obj = i.next();
hashCode = 31*hashCode + (obj==null ? 0 : obj.hashCode());
}

int indexOf(Object o) รีเทิร์นตำแหน่งที่มี o อยู่เป็นตำแหน่งแรก หรือ -1 ถ้า o ไม่ได้อยู่ในลิสต์นี้
boolean isEmpty() รีเทิร์น true ถ้าลิสต์นี้เป็นลิสต์ว่าง
Iterator<E> iterator() รีเทิร์นอิเทอเรเตอร์
int lastIndexOf(Object o) รีเทิร์นตำแหน่งที่มี o อยู่เป็นตำแหน่งสุดท้าย หรือ -1 ถ้า o ไม่ได้อยู่ในลิสต์นี้
ListIterator<E> listIterator() รีเทิร์นลิสต์อิเทอเรเตอร์
ListIterator<E> listIterator(int index) รีเทิร์นลิสต์อิเทอเรเตอร์ โดยให้ตำแหน่งที่สนใจเริ่มจากตำแหน่ง index นี่จะเป็นตำแหน่งที่เมธอด next() รีเทิร์นมาเป็นตำแหน่งแรก ส่วนการเรียก previous() ครั้ง

แรกจากสถานะนี้จะรีเทิร์นสมาชิกตัวที่มีตำแหน่งน้อยกว่านี้หนึ่งตำแหน่ง
E remove(int index) เอาสมาชิกที่ตำแหน่ง index ออกจากลิสต์ รีเทิร์นสมาชิกนั้นออกมา ส่วนสมาชิกตัวอื่นก็เลื่อนมาแทนที่ตำแหน่งของตัวที่ออกไปนี้
boolean remove(Object o) เอาสมาชิกที่ค่าเท่ากับ o (เทียบด้วย equals()) อยู่เป็นตำแหน่งแรกออกจากลิสต์ไป ถ้า o ไม่ได้อยู่ในลิสต์นี้ก็จะไม่มีอะไรเปลี่ยนแปลง รีเทิร์น true ถ้ามีสมาชิกที่มีค่าเท่ากับ o อยู่ในลิสต์จริงๆ
boolean removeAll(Collection<?> c) เอาของที่อยู่ใน c ออกจากลิสต์ไปให้หมด รีเทิร์น true ถ้าลิสต์เกิดการเปลี่ยนแปลง
boolean retainAll(Collection<?> c) เอาสมาชิกที่เหมือนกับสมาชิกใน c (โดยเทียบด้วย equals()) เหลือไว้ นอกนั้นลบทิ้งหมด รีเทิร์น true ถ้ามีสมาชิกถูกเอาออกไปจริงๆ c ห้ามเป็น null ไม่งั้นจะ throw exception
E set(int index, E element) เปลี่ยนสมาชิก ณ ตำแหน่ง index ให้เป็น element รีเทิร์นสมาชิกตัวที่อยู่ก่อนจะถูกเปลี่ยน
int size() รีเทิร์นจำนวนสมาชิกในคอลเลคชั่นนี้ ถ้าจำนวนสมาชิกมากกว่า Integer.MAX_VALUE ให้รีเทิร์น Integer.MAX_VALUE
List<E> subList(int fromIndex, int toIndex) รีเทิร์นลิสต์ย่อย นับรวมตั้งแต่ตำแหน่ง fromIndex จนถึงตำแหน่ง toIndex-1 (ถ้า fromIndex เท่ากับ toIndex เมธอดนี้จะรีเทิร์นลิสต์ว่าง) การเปลี่ยนแปลงในลิสต์ย่อยจะเห็นผลที่ลิสต์ใหญ่ด้วย และการเปลี่ยนแปลงที่ลิสต์ใหญ่ก็จะเห็นผลที่ลิสต์ย่อย ลิสต์ย่อยที่รีเทิร์นมาสามารถใช้เมธอดได้แบบลิสต์ใหญ่ตัวที่เรียกใช้เมธอดนี้ เราสามารถใช้ลิสต์ย่อยในการทำงาน

เฉพาะส่วนได้เช่น list.subList(from, to).clear(); ถ้าลิสต์ตัวใหญ่ถูกเปลี่ยนขนาดหรือถูกทำให้ใช้งานอิเทอเรเตอร์ไม่ได้ผล เมธอดนี้ก็จะใช้ไม่ได้
Object[] toArray() เหมือนกับเมธอดของคอลเลคชั่น
<T> T[] toArray(T[] a) เหมือนกับเมธอดของคอลเลคชั่น

ListIterator

void add(E o) ใส่ o ลงไปในลิสต์
boolean hasNext() รีเทิร์น true ถ้ายังมีสมาชิกของลิสต์ให้ดูอยู่ ในทิศทางที่ไปข้างหน้า (นั่นคือ ถ้าการเรียก next() ครั้งต่อไปรีเทิร์นสมาชิกในลิสต์มา)
boolean hasPrevious() รีเทิร์น true ถ้ายังมีสมาชิกให้ดูในทิศทางย้อนกลับ (นั่นคือ ถ้าการเรียก previous() ครั้งต่อไปรีเทิร์นสมาชิกในลิสต์มา)
E next() รีเทิร์นสมาชิกตัวถัดไปในลิสต์ ถ้าเราเรียก next() แล้ว เรียก previous() สลับกันครั้งต่อครั้ง จะได้สมาชิกตัวเดียวกันเป็นผลลัพธ์ตลอด
int nextIndex() รีเทิร์นค่าดัชนีของสมาชิกตัวที่จะเป็นคำตอบของ next() หรือถ้าอยู่ที่ตำแหน่งสุดท้ายของลิสต์แล้วก็ให้รีเทิร์นขนาดของลิสต์
E previous() รีเทิร์นสมาชิกตัวก่อนในลิสต์

int previousIndex() รีเทิร์นค่าดัชนีของสมาชิกตัวที่จะเป็นคำตอบของ previous() หรือถ้าอยู่ที่ตำแหน่งแรกของลิสต์แล้วก็ให้รีเทิร์น-1
void remove() เอาสมาชิกตัวที่พึ่งเป็นผลลัพธ์ของ next() หรือ previous() ออกจากลิสต์ เมธอดนี้ใช้สลับกับ next() หรือ previous()ได้ครั้งต่อครั้งเท่านั้น จะต้องไม่มีการเรียกเมธอด ListIterator.add() ระหว่างการเรียก next() หรือ previous() กับเมธอดนี้
void set(E o) เอา o แทนที่สมาชิกตัวท้ายสุดที่พึ่งถูกรีเทิร์นเป็นคำตอบของ next() หรือ previous() นอกจากนี้จะต้องไม่มีการเรียกเมธอด ListIterator.add() และ ListIterator.remove() ระหว่างการเรียก next() หรือ previous() กับเมธอดนี้

รูปที่ 2 เมธอดต่างๆของ จาวาคอลเลคชั่น