ARRAYLIST SOLUTIONS

Solution 1:

```
public boolean isMonotonic(ArrayList<Integer> A) {
        boolean inc = true;
        boolean dec = true;
        for (int i=0; i<A.size()-1; i++) {
            if (A.get(i) > A.get(i+1))
                inc = false;
            if (A.get(i) < A.get(i+1))
                dec = false;
        }

        return inc || dec;
    }
```

Solution 2:

```
public ArrayList<Integer> findLonely(ArrayList<Integer> nums) {
        Collections.sort(nums);
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=1; i < nums.size()-1; i++) {
              if (nums.get(i-1) + 1 < nums.get(i) && nums.get(i) + 1 < nums.get(i+1))
{
                 list.add(nums.get(i));
            }
        }
        if (nums.size() == 1) {
            list.add(nums.get(0));
        }
        if (nums.size() > 1) {
            if (nums.get(0) + 1 < nums.get(1)) {
                list.add(nums.get(0));
            }
            if (nums.get(nums.size()-2) + 1 < nums.get(nums.size()-1)) {
```

```
                list.add(nums.get(nums.size()-1));
            }
        }
        return list;
    }
```

Solution 3 :

```
public int mostFrequent(ArrayList<Integer> nums, int key) {
        int[]result = new int[1000];

        for(int i=0; i<nums.size()-1; i++){
            if(nums.get(i) == key){
                result[nums.get(i+1)-1]++;
            }
        }

        int max = Integer.MIN_VALUE;
        int ans = 0;

        for(int i=0; i<1000; i++){
            if(result[i] > max){
                max = result[i];
                ans = i+1;
            }
        }

        return ans;
    }
```

Solution 4 :

Approach 1 (Iterative)

We can see that if we separate odd and even numbers then there is no possibility that even numbers will violate the rule with odd numbers and vice versa. Now we have to arrange even and odd numbers in such a way that they do not violate rules with themselves. For doing so

first let's say we have a beautiful arraylist of size n and we want to make n+1 size of arraylist so what we do is first put all odd numbers that lie within 1 to n+1 and then even(WE can do even then odd also) . Beautiful arraylist has the property that if we multiply any number with arraylist then it still remains beautiful or if we add or subtract any number from arraylist then it still remains beautiful. For only obtaining even number from n size arraylist we do 2* num and for obtaining odd size arraylist we do 2 * n-1 .

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> ans = new ArrayList<>();
        ans.add(1);

        for(int i=2;i<=n;i++){
            ArrayList<Integer>temp=new ArrayList<>();
            for(Integer e:ans){
                if(2*e<=n)temp.add(e*2);
            }
            for(Integer e:ans){
                if(2*e-1<=n)temp.add(e*2-1);
            }

            ans=temp;
        }

        return ans;
    }
```

Approach 2 (Divide & Conquer)

Let's start from a simple 3 numbers case: (1, 2, 3) -> the only thing we need to do is move 2 out of 1 and 3 -> (1, 3, 2).

Then what if the case is (1, 5, 9) which has increment = 4? It's the same thing -> move 3 out of 1 and 5 -> (1, 9, 5).

Now, what if the case is (1, 3, 5, 7, 9) ? With the odd + even or divide + conquer idea in mind, we can simply divide it to (1, 5, 9) and (3, 7). Since no change is needed for the 2 numbers case, after following the above steps, we can conquer them to (1, 9, 5, 3, 7).

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> res = new ArrayList<>();
        divideConque(1, 1, res, n);
        return res;
    }
```

```
    private void divideConque(int start, int increment, ArrayList<Integer> res, int
n) {
        if (start + increment > n) {
            res.add(start);
            return;
        }
        divideConque(start, 2 * increment, res, n);
        divideConque(start + increment, 2 * increment, res, n);
    }
```