CS 61B Spring 2025

Iterators, Asymptotics

Exam-Level 04: February 17, 2025

1 Iterator of Iterators

Implement an **IteratorOfIterators** which takes in a **List** of **Iterator**s of **Integers** as an argument. The first call to **next()** should return the first item from the first iterator in the list. The second call should return the first item from the second iterator in the list. If the list contained **n** iterators, the **n+1**th time that we call **next()**, we would return the second item of the first iterator in the list.

Note that if an iterator is empty in this process, we continue to the next iterator. Then, once all the iterators are empty, **hasNext** should return **false**. For example, if we had 3 **Iterators** A, B, and C such that A contained the values **[1, 3, 4, 5]**, B was empty, and C contained the values **[2]**, calls to **next()** for our **IteratorOfIterators** would return **[1, 2, 3, 4, 5]**.

```
public class IteratorOfIterators ____________________________________________ {
    private List<Iterator<Integer>> iterators;
    private int curr;
    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (_____________________________________________) {
            if (_____________________________________________) {
                _____________________________________________;
            }
        }
        curr = 0;
    }
    @Override
    public boolean hasNext() {
        return _____________________________________________;
    }
    @Override
    public Integer next() {
        if (!hasNext()) { throw new NoSuchElementException(); }
        Iterator<Integer> currIterator = _____________________________________________;
        int result = _____________________________________________;
        if (_____________________________________________) {
            _____________________________________________;
            if (iterators.isEmpty()) {
                _____________________________________________;
            }
        } else {
            curr = _____________________________________________;
        }
        return result;
    }
}
```

2 Asymptotics

(a) Say we have a function **findMax** that iterates through an unsorted int array one time and returns the maximum element found in that array. Give the tightest lower and upper bounds ($\Omega(\cdot)$ and $O(\cdot)$) of **findMax** in terms of **N**, the length of the array. Is it possible to define a $\Theta(\cdot)$ bound for **findMax**?

(b) Give the worst case and best case runtime in terms of M and N. Assume **ping** runs in $\Theta(1)$ and returns an **int**.

```
for (int i = N; i > 0; i--) {
    for (int j = 0; j <= M; j++) {
        if (ping(i, j) > 64) { break; }
    }
}
```

(c) Below we have a function that returns **true** if every **int** has a duplicate in the array, and **false** if there is any unique int in the array. Assume **sort(array)** is in $\Theta(N \log N)$ and returns **array** sorted.

```
public static boolean noUniques(int[] array) {
    array = sort(array);
    int N = array.length;
    for (int i = 0; i < N; i += 1) {
        boolean hasDuplicate = false;
        for (int j = 0; j < N; j += 1) {
            if (i != j && array[i] == array[j]) {
                hasDuplicate = true;
            }
        }
        if (!hasDuplicate) return false;
    }
    return true;
}
```

Give the worst case and best case runtime where $N =$ **array.length**.