Disclaimer: These notes may be distributed outside this class only with the permission of the Instructor.

3.1 Array Based Collections

Listing 1: Bag Class

```java
import java.util.Iterator;

/**
 * The Bag class represents a collection of generic items.
 * It supports insertion and iterating over the items in arbitrary order.
 */
public class Bag<E> implements Iterable<E> {
    protected E[] items; // array of items
    protected int N = 0; // number of items in the bag
    protected int capacity = 10; // capacity of the bag

    /**
     * Initializes an empty bag.
     */
    Bag() {
        items = (E[]) new Object[capacity];
    }

    /**
     * Returns an iterator that iterates through the items in the bag
     * @return an iterator that iterates through the items in the bag
     */
    public Iterator<E> iterator() {
        return new BagIterator();
    }

    /**
     * The iterator implementation
     */
    private class BagIterator implements Iterator<E> {
        private int i = 0;

        public boolean hasNext() {
            return i < N;
        }

        public void remove() {
            System.out.println("to be implemented.");
        }

        public E next() {
            if(!hasNext()) {
                return null;
            }
            return items[i++];
        }
    }

    /**
     * Insert new items into the bag
     * @param item the new item to be inserted.
     */
```
public void insert(E item)
{
    if (N == capacity) {
        resize();
    }
    items[N] = item;
    N++;
}

/**
 * Returns an item by index
 * @param index is the item index
 */
public E get(int index)
{
    return items[index];
}

/**
 * size of the bag
 * @return size the number of items in the bag.
 */
public int size()
{
    return N;
}

/**
 * if the bag contains a given item?
 * @return true if bag contains the item. false otherwise
 */
public boolean contains(E item)
{
    for (int i = 0; i < N; ++i) {
        if (items[i].equals(item)) return true;
    }
    return false;
}

/**
 * is the bag empty?
 * @return true if bag is empty. false otherwise
 */
public boolean isEmpty()
{
    return N == 0;
}

/**
 * Resize the bag when capacity is not enough
 */
protected void resize()
{
    capacity *= 2;
    int index = 0;
    E[] temp = (E[]) new Object[capacity];
    for (E e : items) {
        temp[index++] = e;
    }
    N = index;
    items = temp;
}

/**
 * unit test for bag
 */
public static void main(String[] args)
{
    Bag<Integer> bag = new Bag();
    for (int i = 1; i <= 20; i++) {
        bag.insert(i);
    }
}
For this implementation of the Bag class, the time complexity of the size, isEmpty, insert, and get methods are $O(1)$, which means that the running time of those methods do not depend on the input size. It always takes constant amount of time. The time complexity of the contains method is $O(n)$, which means that for large enough input sizes the running time increases linearly with the size of the input, as shown in the Figure 3.1.

![Graph showing processing time increases as the number of items in the bag increases](image-url)