

Lecture 3:

Lecturer: Anwar Mamat

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

```
1 import java.util.Iterator;
2 /**
3  * The Bag class represents a collection of generic items.
4  * It supports insertion and iterating over the items in arbitrary order.
5  */
6 public class Bag<E> implements Iterable<E>
7 {
8     protected E[] items; //array of items
9     protected int N = 0; //number of items in the bag
10    protected int capacity = 10; //capacity of the bag
11
12    /**
13     * Initializes an empty bag.
14     */
15    Bag()
16    {
17        items = (E[]) new Object[capacity];
18    }
19    /**
20     * Returns an iterator that iterates through the items in the bag
21     * @return an iterator that iterates through the items in the bag
22     */
23    public Iterator<E> iterator() {
24        return new BagIterator();
25    }
26    /**
27     * The iterator implementation
28     */
29    private class BagIterator implements Iterator<E> {
30        private int i = 0;
31        public boolean hasNext() {
32            return i < N;
33        }
34        public void remove() {
35            System.out.println("to_be_implemented.");
36        }
37        public E next() {
38            if(!hasNext()) {
39                return null;
40            }
41            return items[i++];
42        }
43    }
44 }
45
46 /**
47  * Insert new items into the bag
48  * @param item the new item to be inserted.
49  */
```

```

50     public void insert(E item)
51     {
52         if(N == capacity){
53             resize();
54         }
55         items[N] = item;
56         N++;
57     }
58
59     /**
60     * Returns an item by index
61     * @param index is the item index
62     */
63     public E get(int index)
64     {
65         return items[index];
66     }
67
68     /**
69     * size of the bag
70     * @return size the number of items in the bag.
71     */
72     public int size(){
73         return N;
74     }
75
76     /**
77     * if the bag contains a given item?
78     * @return true if bag contains the item. false otherwise
79     */
80     public boolean contains(E item)
81     {
82         for(int i = 0; i < N; ++i){
83             if(items[i].equals(item)) return true;
84         }
85         return false;
86     }
87     /**
88     * is the bag empty?
89     * @return true if bag is empty. false otherwise
90     */
91     public boolean isEmpty()
92     {
93         return N == 0;
94     }
95     /**
96     * Resize the bag when capacity is not enough
97     */
98     protected void resize(){
99         capacity *= 2;
100        int index =0;
101        E[] temp = (E[]) new Object[capacity];
102        for(E e: items){
103            temp[index++] = e;
104        }
105        N = index;
106        items = temp;
107    }
108
109    /**
110    * unit test for bag
111    */
112    public static void main(String[ ] args)
113    {
114        Bag<Integer> bag = new Bag();
115        for(int i = 1; i <= 20; i++){
116            bag.insert(i);
117        }

```

```
118
119     /*for(int i = 0; i < bag.size(); i++){
120         System.out.println(bag.get(i));
121     }*/
122
123     for(Integer i: bag){
124         System.out.print(i+", ");
125     }
126 }
127 }
```

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.

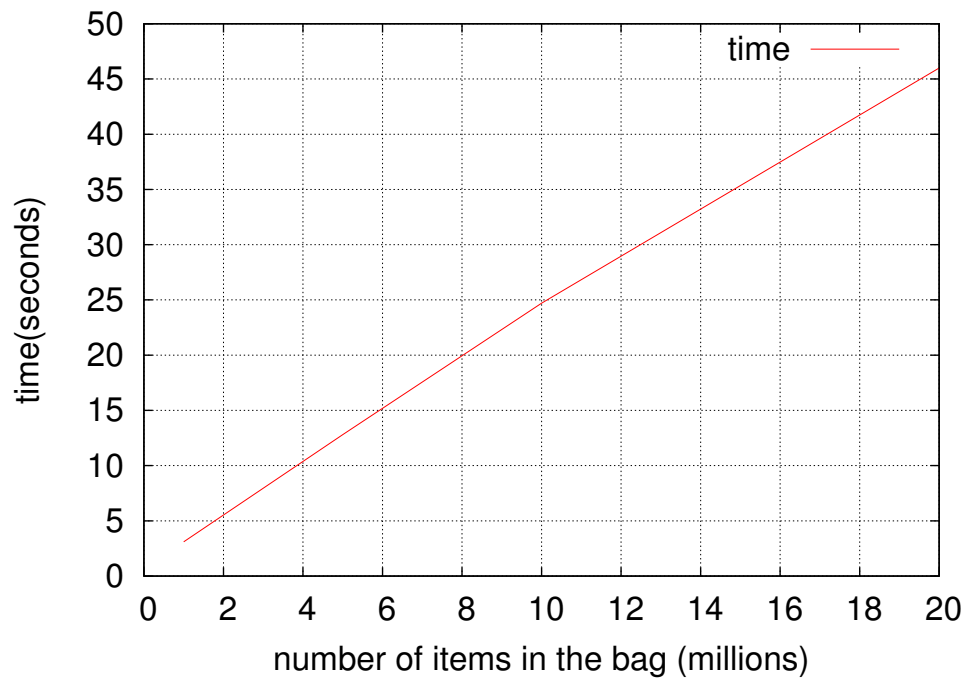


Figure 3.1: Processing time increases as the number of items in the bag increases