CMSC 132, Object-Oriented Programming II **Summer 2018**

Lecture 4:

Lecturer: Anwar Mamat

As we learned from the previous lecture, the time complexity of "contains" method in the Bag class is O(n). This is because, in worst case, we have to check every item in the items array to know if a given item exists in the array. For example, if we look for 6 from the array given in Figure 4.1, we will have to check every single item in the array. On average, we compare $\frac{N}{2}$ times, where N is the length of the array. Can we do better?

Figure 4.1: Unsorted Array

If the array is sorted, then we can use binary search, which has the time complexity of $log(N)$. For example, to find 9 from the sorted array in Figure 4.2, we first compare 9 with 5, the item in middle of the array. Because 9 is greater than 5, it means 9 can only exist in the right side of 5. In this way, we can cut half of the array after each comparison. $log(n)$ is a huge improvement.

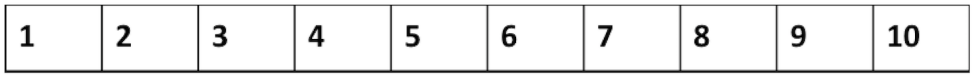

Figure 4.2: Sorted Array

4.1 Sorted Bag

In this section, we extend the Bag class such that the extended bag, SortedBag, inserts the items into bah in sorted order. Because the items are sorted, we can use binary search for the contains method. To sort the items, we should be able to compare them to each other. Therefore, items in SortedBag must implement Comparable interface.

Listing 1: Sorted Bag Class

```
1  public class SortedBag<E extends Comparable> extends Bag<E> {
2          public SortedBag(){
3                  //We cannot use the Object array in the Bag because Object
4                  //cannot be cast into Comparable.
5                  items = (E[])new Comparable[capacity];
6          }
7          /**
8          *       doubles the size of items array.
9          *       We override this method because we
10         *       create Comparable[ ]
```

```
11          */
12          @Override
13          protected void resize(){
14                  capacity *= 2;
15                  E[] temp = (E[])new Comparable[capacity];
16                  for(int i =0; i <N; i++){
17                          temp[i] = items[i];
18                  }
19                  items = temp;
20          }
21          /**
22          *       Insert an item in sorted order.
23          */
24          @Override
25          public void insert(E item){
26                  if(N == capacity){
27                          resize();
28                  }
29                  if(N == 0){
30                          items[0] = item;
31                          N++;
32                          return;
33                  }
34                  int index = N-1;
35                  while(items[index].compareTo(item) >0){
36                          items[index+1] = items[index];
37                          index--;
38                          if(index <0) break;
39                  }
40                  items[++index] = item;
41                  N++;
42          }
43          /**
44          *       binary search from the bag
45          */
46          public boolean contains(E item){
47                  int m = 0;
48                  int s = 0;
49                  int t = N-1;
50                  while(s <= t){
51                          m = (t+s)/2;
52                          if(items[m].equals(item)) return true;
53                          if(items[m].compareTo(item) > 0 ){
54                                  t = m - 1;
55                          }else {
56                                  s = m+1;
57                          }
58                  }
59                  return false;
60          }
61  }
```