

Andrew Walters
CSC 2053
Youtube Comedy Slam Project

The YoutubeComedySlam, BinarySearchST, and Item classes were all implemented successfully and testing showed that they operated as expected. When the instructor provided dataset was used as input, the output matched the instructor's output. Below see a complete listing of the tests performed on the system along with a screenshot of the testing. See the appendix for a complete listing of the code.

```
> run YoutubeComedySlam test.txt
Total number of youtube videos in the running 78.0
Average number of wins 2.782051282051282
Top winner = hI6D4bynjW4 won 10 times.
>
> run YoutubeComedySlam comedy_comparisons.test
Total number of youtube videos in the running 3997.0
Average number of wins 56.440580435326495
Top winner = vZxL0jcVu9s won 8127 times.
>
> run YoutubeComedySlam comedy_comparisons.train
Total number of youtube videos in the running 14429.0
Average number of wins 63.273199805946355
Top winner = C8IJnUM0yQo won 33653 times.
```

The screenshot shows the DrJava IDE with the file `C:\Users\Andrew\Google Drive\Villanova\Algorithms III\CSC2053_Project2_YoutubeComedySlam\YoutubeComedySlam.java` open. The editor displays the following Java code:

```
1 import java.util.Scanner;
2 import java.io.*;
3
4 public class YoutubeComedySlam {
5
6     private static String left = ",left";
7
8     public static void main(String args[]) throws IOException {
```

Below the editor, the **Console** tab is active, showing the output of running the program. The output includes the following text:

```
Welcome to DrJava. Working directory is C:\Users\Andrew\Google Drive\Villanova\Algorithms III\CSC2053_Project2_YoutubeComedySlam
> run YoutubeComedySlam test.txt
Total number of youtube videos in the running 78.0
Average number of wins 2.782051282051282
Top winner = h16D4bynjW4 won 10 times.
> run YoutubeComedySlam comedy_comparisons.test
Total number of youtube videos in the running 3597.0
Average number of wins 56.440580435326495
Top winner = v2xL0jcVu9s won 8127 times.
> run YoutubeComedySlam comedy_comparisons.train
Total number of youtube videos in the running 14429.0
Average number of wins 63.273199805946355
Top winner = C81JnUM0y0o won 33653 times.
>
```

The status bar at the bottom indicates the file path and the line number 65/32.

```

1  /*****
2  * Compilation:  javac YoutubeComedySlam.java
3  * Execution:   java YoutubeComedySlam input
4  * Dependencies: java.io.* java.util.Scanner BinarySearchST.java Item.java
5  * Data files:  http://algs4.cs.princeton.edu/31elementary/tinyST.txt
6  *
7  * Reads data from UCI machine learning repo into BinarySearchST and
8  * determines video with the most votes, average num of votes per video
9  *
10 *****/
11
12 import java.util.Scanner;
13 import java.io.*;
14
15 public class YoutubeComedySlam {
16
17     private static String left = ",left";
18
19     public static void main(String args[]) throws IOException {
20
21         String vidL,vidR,winner;
22         int tempi;
23
24         BinarySearchST<String,Integer> vids = new BinarySearchST<String,Integer>();
25
26         Scanner scan = null;
27
28         try {
29             scan = new Scanner(new BufferedReader(new FileReader(args[0])));
30             scan.useDelimiter(",");
31
32             while(scan.hasNext()) {
33
34                 vidL = scan.next();
35                 vidR = scan.next();
36
37                 //determine winner
38                 winner = scan.nextLine();
39
40                 if(winner.equals(left)) {
41                     winner = vidL;
42                 }
43                 else {
44                     winner = vidR;
45                 }
46
47                 //determine new value of winner
48                 if(vids.contains(winner)) {
49                     tempi = vids.get(winner) + 1;
50                 }
51                 else {
52                     tempi = 1;
53                 }
54
55                 //create winner or increment winner's value
56                 vids.put(winner,tempi);
57
58             } //end loop
59         } //end try
60         finally {
61             if(scan != null) {
62                 scan.close();
63             }
64         } //end finally
65
66         //iterate through vids to finds stats
67         Iterable<String> list = vids.keys();
68
69         String max = vids.min();
70         double total = 0;

```

s\Andrew\Google Drive\Villanova\Algorithms III\CSC2053_Project2_YoutubeComedySlam\YoutubeComedySlam

```
71         double wins = 0;
72         double av;
73
74         for(String key: list) {
75             total++;
76             wins+=vids.get(key);
77             if(vids.get(max) < vids.get(key)) {
78                 max = key;
79             }
80         } //end loop
81
82         av = wins/total;
83         System.out.println("Total number of youtube videos in the running " +
84 total);
85         System.out.println("Average number of wins " + av);
86         System.out.println("Top winner = " + max + " won " + vids.get(max) + "
87 times.");
88     } //end main
89 } //end class
```

```

1  /*****
2  * Compilation:  javac BinarySearchST.java
3  * Execution:    java BinarySearchST
4  * Dependencies: StdIn.java StdOut.java Item.java
5  * Data files:   http://algs4.cs.princeton.edu/31elementary/tinyST.txt
6  *
7  * Symbol table implementation with binary search in an ordered array.
8  *
9  *
10 *****/
11
12 import java.util.NoSuchElementException;
13
14 public class BinarySearchST<Key extends Comparable<Key>, Value> {
15     private static final int INIT_CAPACITY = 2;
16     private Object[] objects;
17     private int N = 0;
18
19     // create an empty symbol table with default initial capacity
20     public BinarySearchST() {
21         this(INIT_CAPACITY);
22     } //end constructor
23
24     // create an empty symbol table with given initial capacity
25     public BinarySearchST(int capacity) {
26         objects = new Object[capacity];
27     } //end constructor
28
29     // resize the underlying arrays
30     private void resize(int capacity) {
31         assert capacity >= N;
32         Object[] tempo = new Object[capacity];
33
34         for (int i = 0; i < N; i++) {
35             tempo[i] = objects[i];
36         } //end loop
37
38         objects = tempo;
39     }
40
41     // is the key in the table?
42     public boolean contains(Key key) {
43         return get(key) != null;
44     }
45
46     // number of key-value pairs in the table
47     public int size() {
48         return N;
49     }
50
51     // is the symbol table empty?
52     public boolean isEmpty() {
53         return size() == 0;
54     }
55
56     // return the value associated with the given key, or null if no such key
57     public Value get(Key key) {
58         if (isEmpty()) {
59             return null;
60         }
61         int i = rank(key);
62         if (i >= N) {
63             return null;
64         }
65         Item<Key, Value> temp = (Item) objects[i];
66         if (i < N && (temp.getKey()).compareTo(key) == 0) {
67             return temp.getValue();
68         }
69         return null;
70     }

```

```

71
72 // return the number of keys in the table that are smaller than given key
73 public int rank(Key key) {
74     int lo = 0, hi = N-1;
75     Item<Key,Value> temp;
76     while (lo <= hi) {
77         int m = lo + (hi - lo) / 2;
78         temp = (Item)objects[m];
79         int cmp = key.compareTo(temp.getKey());
80         if (cmp < 0) hi = m - 1;
81         else if (cmp > 0) lo = m + 1;
82         else return m;
83     }
84     return lo;
85 }
86
87
88 // Search for key. Update value if found; grow table if new.
89 public void put(Key key, Value val) {
90     if (val == null) { delete(key); return; }
91
92     int i = rank(key);
93     // key is already in table
94     if (i < N) {
95         Item<Key,Value> temp = (Item)objects[i];
96         if ((temp.getKey()).compareTo(key) == 0) {
97             temp.setValue(val);
98             return;
99         }
100     }
101
102     // insert new key-value pair
103     if (N == objects.length) {
104         resize(2*objects.length);
105     }
106
107     for (int j = N; j > i; j--) {
108         objects[j] = objects[j-1];
109     } //end loop
110
111     objects[i] = new Item<Key,Value>(key,val);
112     N++;
113
114     assert check();
115 } //end put
116
117
118 // Remove the key-value pair if present
119 public void delete(Key key) {
120     if (isEmpty()) return;
121
122     // compute rank
123     int i = rank(key);
124     Item<Key,Value> temp = (Item)objects[i];
125
126     // key not in table
127     if (i == N || (temp.getKey()).compareTo(key) != 0) {
128         return;
129     }
130
131     for (int j = i; j < N-1; j++) {
132         objects[j] = objects[j+1];
133     } //end loop
134
135     N--;
136     objects[N] = null; // to avoid loitering
137
138     // resize if 1/4 full
139     if (N > 0 && N == objects.length/4) {
140         resize(objects.length/2);

```

```

141         }
142
143         assert check();
144     } //end delete
145
146     // delete the minimum key and its associated value
147     public void deleteMin() {
148         if (isEmpty()) throw new NoSuchElementException("Symbol table underflow
error");
149         delete(min());
150     }
151
152     // delete the maximum key and its associated value
153     public void deleteMax() {
154         if (isEmpty()) throw new NoSuchElementException("Symbol table underflow
error");
155         delete(max());
156     }
157
158
159     /*****
160     *   Ordered symbol table methods
161     *****/
162     public Key min() {
163         if (isEmpty()) return null;
164         Item<Key,Value> temp = (Item)objects[0];
165         return temp.getKey();
166     }
167
168     public Key max() {
169         if (isEmpty()) return null;
170         Item<Key,Value> temp = (Item)objects[N-1];
171         return temp.getKey();
172     }
173
174     public Key select(int k) {
175         if (k < 0 || k >= N) {
176             return null;
177         }
178
179         Item<Key,Value> temp = (Item)objects[k];
180         return temp.getKey();
181     }
182
183     public Key floor(Key key) {
184         int i = rank(key);
185         Item<Key,Value> temp = (Item)objects[i];
186
187         if (i < N && key.compareTo(temp.getKey()) == 0) {
188             return temp.getKey();
189         }
190         if (i == 0) {
191             return null;
192         }
193         else {
194             Item<Key,Value> tempDec = (Item)objects[i-1];
195             return tempDec.getKey();
196         }
197     } //end floor
198
199     public Key ceiling(Key key) {
200         int i = rank(key);
201         Item<Key,Value> temp = (Item)objects[i];
202
203         if (i == N) {
204             return null;
205         }
206         else {
207             return temp.getKey();
208         }

```

```

209     } //end ceiling
210
211     public int size(Key lo, Key hi) {
212         if (lo.compareTo(hi) > 0) {
213             return 0;
214         }
215         if (contains(hi)) {
216             return rank(hi) - rank(lo) + 1;
217         }
218         else {
219             return rank(hi) - rank(lo);
220         }
221     } //end size
222
223     public Iterable<Key> keys() {
224         return keys(min(), max());
225     }
226
227     public Iterable<Key> keys(Key lo, Key hi) {
228         Queue<Key> queue = new Queue<Key>();
229
230         if (lo == null && hi == null) {
231             return queue;
232         }
233         if (lo == null) {
234             throw new NullPointerException("lo is null in keys()");
235         }
236         if (hi == null) {
237             throw new NullPointerException("hi is null in keys()");
238         }
239         if (lo.compareTo(hi) > 0) {
240             return queue;
241         }
242
243         Item<Key, Value> temp;
244         for (int i = rank(lo); i < rank(hi); i++) {
245             temp = (Item)objects[i];
246             queue.enqueue(temp.getKey());
247         }
248
249         if (contains(hi)) {
250             temp = (Item)objects[rank(hi)];
251             queue.enqueue(temp.getKey());
252         }
253
254         return queue;
255     }
256
257
258     /*****
259     * Check internal invariants
260     *****/
261
262     private boolean check() {
263         return isSorted() && rankCheck();
264     }
265
266     // are the items in the array in ascending order?
267     private boolean isSorted() {
268         int i = 0;
269         Item<Key, Value> tempDec;
270         Item<Key, Value> temp = (Item)objects[i];
271
272         for (i = 1; i < size(); i++) {
273             tempDec = temp;
274             temp = (Item)objects[i];
275             if ((temp.getKey()).compareTo(tempDec.getKey()) < 0) {
276                 return false;
277             }
278         } //end loop

```



```

279         return true;
280     } //end isSorted
281
282     // check that rank(select(i)) = i
283     private boolean rankCheck() {
284         for (int i = 0; i < size(); i++) {
285             if (i != rank(select(i))) {
286                 return false;
287             }
288         } //end loop
289
290         Item<Key,Value> temp;
291         for (int i = 0; i < size(); i++) {
292             temp = (Item)objects[i];
293             if ((temp.getKey()).compareTo(select(rank(temp.getKey()))) != 0) {
294                 return false;
295             }
296         } //end loop
297         return true;
298     } //end rankCheck
299
300
301     /*****
302     *   Test client
303     *****/
304     public static void main(String[] args) {
305         BinarySearchST<String, Integer> st = new BinarySearchST<String, Integer>();
306         for (int i = 0; !StdIn.isEmpty(); i++) {
307             String key = StdIn.readString();
308             if (key.compareTo("exit")==0) {
309                 break;
310             }
311             st.put(key, i);
312         }
313         for (String s : st.keys())
314             StdOut.println(s + " " + st.get(s));
315     }
316 }

```

```

1  /*****
2  *  Compilation:  javac Item.java
3  *  Execution:   java Item
4  *  Dependencies: none
5  *  Data files:  http://algs4.cs.princeton.edu/31elementary/tinyST.txt
6  *
7  *  Item class stores a key and value, each a parameterized type. The value
8  *  can be modified after being initiated but not the key.
9  *
10 *****/
11
12 public class Item<Key extends Comparable<Key>, Value> {
13
14     private Key key;
15     private Value value;
16
17     public Item(Key k, Value v) {
18         key = k;
19         value = v;
20     } //end constructor
21
22     public Key getKey() {
23         return key;
24     } //end getKey
25
26     public Value getValue() {
27         return value;
28     } //end getValue
29
30     public void setValue(Value v) {
31         value = v;
32     } //end setValue
33
34     public static void main(String[] args) {
35         Item<String, Integer> test = new Item<String, Integer> ("Andrew",0);
36         System.out.println(test.getKey() + ": " + test.getValue());
37     } //end main
38
39 } //end class

```