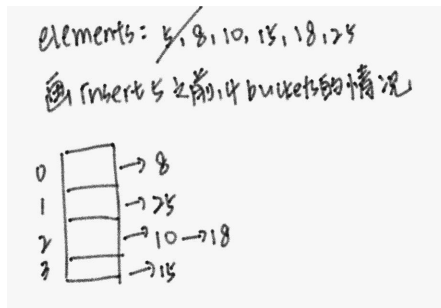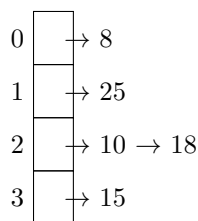## 1   Warmup (Spring 2015 MT2: 1c)

Draw the External Chaining Hash Set that results if we insert 5. As part of this insertion, you should also resize from 4 buckets to 8 (in other words, the implementer of this data structure seems to be resizing when the load factor reaches 1.5). Assume that were using the default hashCode for integers, which simply returns the integer itself.

2   Hashtable Runtimes (Fall 2016 MT2: Q3)   hashset

Consider a hash table that uses external chaining and also keeps track of the number of keys that it contains. It stores each key at most once; adding a key a second time has no effect. It takes the steps necessary to ensure that the number of keys is always less than or equal to twice the number of buckets (i.e., that the load factor is ≤ 2). Assume that its hash function and comparison of keys take constant time. All bounds should be a function of N, the number of elements in the table.

1. Give $\Theta()$ bounds on the worst-case times of adding an element to the table when the load factor is 1 and when it is exactly 2 <u>before the addition</u>.

   the worst case should be all elements are clustered together in a single bucket

   Bound for load factor 1:   O(N)   although the answer is correct, we should now consider resizing, which takes Big theta of N time

   Bound for load factor 2:   O(N)

2. Assume that the hashing function is so good that it always evenly distributes keys among buckets. What now are the bounds on the worst-case time of <u>adding an element</u>?

   Bound for load factor 1:   O(1)

   Bound for load factor 2:   O(N)   this is because of resizing

3. Making no assumption about the goodness of the hashing function, suppose that instead of using linked lists for the buckets, we use some kind of binary search tree that somehow keeps itself bushy. What bound can you place on the worst-case time for testing to see if an item is in the table?

   Bound: the worst case is still all elements are clustered in a single bucket and retrieving an element from a BST should take O(log N) time

4. Using the same representation as in part (c), but with a very good hash function, as in part (b), what bound can you place on the worst-case time for testing to see if an item is in the table?

   Bound: O(1)

3   Zubat (Summer 2016 MT2: Q3)

Consider the following classes and their hashcodes and equality definitions. There is a problem with each hashCode() method below (correctness, distribution, efficiency). Provide a one sentence explanation. Do not list more than one problem. Assume there are no problems with the correctness of equals; any code for handling casting is omitted for space.

1. DynamicString

```
1    class DynamicString {
2        ArrayList<Character> vals;
3        public int hashCode() {
4            int h = 0;
5            for (int i = 0; i < vals.size(); i++) {
6                h = 31 * h + vals.get(i);
7            }
8            return h;
9        }
10
11       public boolean equals(Object o) {
12           DynamicString d = (DynamicString) o;
13           return vals.equals(d.vals);
14       }
15   }
```

Problem(s), if any: **No problems.** (This one is done for you as an example using Javas String::hashCode.)

2. PokeTime

```
1    class PokeTime {
2        int startTime;
3        int duration;
4
5        public int getCurrentTime() {
6            // Gets the current system clock time
7        }
8
9        public int hashCode() {
10           return 1021 * (startTime + 1021
11                   * duration + getCurrentTime());
12       }
13
14       public boolean equals(Object o) {
15           PokeTime p = (PokeTime) o;
16           return p.startTime == startTime
17                   && p.duration == duration;
18       }
19   }
```

problem: correctness
this hashCode violates the deterministic property of a valid hashCode, because it depends on the current time, which keeps changing

Problem(s), if any:

3. Phonebook

```
1    class Phonebook {
2        List<Human> humans;
3
4        public int hashCode() {
5            int h = 0;
6            for (Human human : humans) {
7                // Assume Human hashcode is correct
8                h = (h + human.hashCode()) % 509;
9            }
10           return h;
11       }
12
13       public boolean equals(Object o) {
14           Phonebook p = (Phonebook) o;
15           return p.humans.equals(humans);
16       }
17   }
```

problem: distribtion
it is too clustered

Problem(s), if any:

4. Person

```
1   class Person {
2       Long id;
3       String name;
4       Integer age;
5
6       public int hashCode() {
7           return id.hashCode() + name.hashCode()
8                   + age.hashCode();
9       }
10
11      public boolean equals(Object o) {
12          Person p = (Person) o;
13          return p.id == id;
14      }
15  }
```

Problem(s), if any: correctness
                      equals only considers id, however, hashCode depends on other two fields

5. DblCharSeq

```
1   class DblCharSeq {
2       char[] seq1;
3       char[] seq2;
4
5       public int hashCode() {
6           int h = 0;
7           for (char c1 : seq1) {
8               for (char c2 : seq2) {
9                   h = 31 * (31 * h + c1) + c2;
10              }
11          }
12          return h;
13      }
14
15      public boolean equals(Object o) {
16          DblCharSeq d = (DblCharSeq) o;
17          return Arrays.equals(seq1, d.seq1)
18                  && Arrays.equals(seq2, d.seq2);
19      }
20  }
```

Problem(s), if any: efficiency
                      the computation of hashCode is too complex