CS 61BL Summer 2021

Hashing

Quiz 9: Wednesday July 28, 2021

1 Hashing

(a) Imagine we have the following class:

```
1  public class Course {
2      public final int CCN;
3      public final String instructor;
4      public Student[] students;
5      public int audited; //when the course was last audited
6      public Course(int CCN, Student[] initial) {
7          this.CCN = CCN;
8          this.students = initial;
9          this.instructor = "Sohum";
10     }
11     //implementation
12     public void audit() {
13         this.audited = System.currentTimeMillis();
14         //implementation
15     }
16     public void addStudent(Student s) {
17         //implementation
18     }
19 }
```

Which of the following hashing functions for the Course class are **valid**?

A)

```
1  @Override
2  public int hashCode() {
3      return CCN;
4  }
```

B)

```
1  @Override
2  public int hashCode() {
3      return this.students.length;
4  }
```

C)

```
1  @Override
2  public int hashCode() {
3      return this.audited;
4  }
```

D)

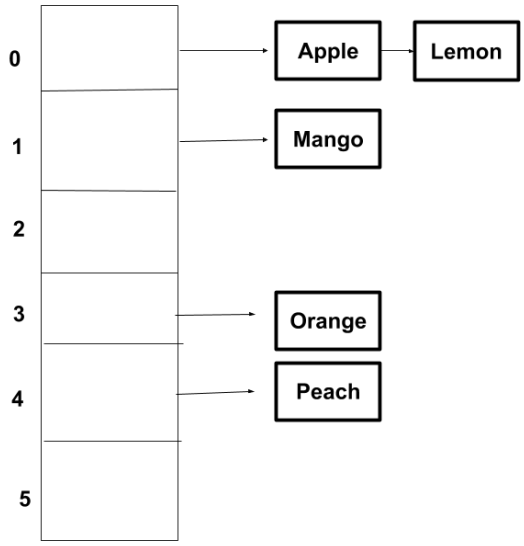
```
@Override
public int hashCode() {
    return 5;
}
```

E)

```
@Override
public int hashCode() {
    return getNumericValue(this.instructor.charAt(0));
}
```

[ ] A
[ ] B
[ ] C
[ ] D
[ ] E

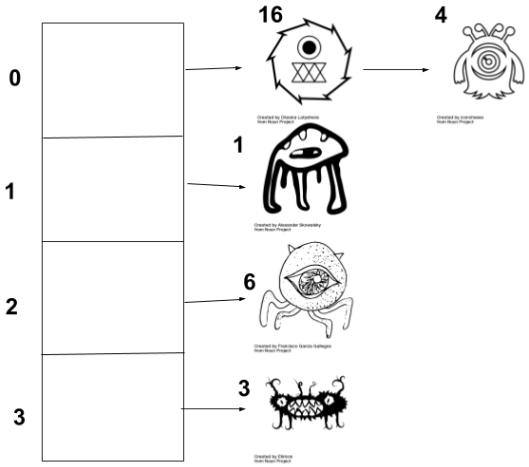
(b) We have the below external chaining HashSet.



If the load factor is 1.25, how many more insertions can we make before we will resize? **Do not** include the insertion that will begin with the resize.

( ) 1
( ) 2
( ) 3
( ) 4
( ) 5
( ) 6

(c) Suppose we have the following MonsterHashTable.



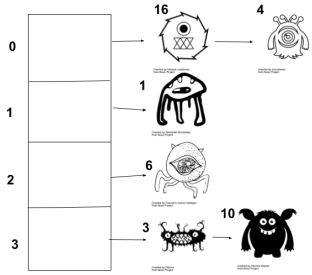
The number to the upper left of the monster is their hashCode.

Part One:

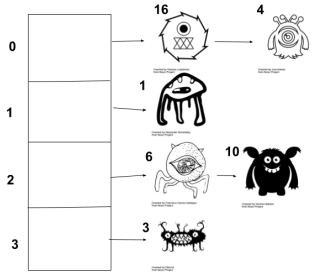
Suppose we want to insert the element:

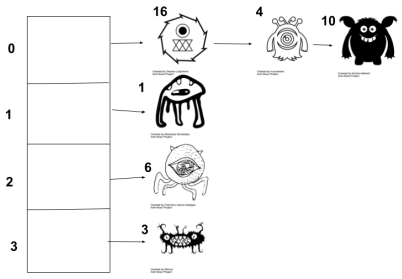
Which of the following would correctly mirror the state after inserting the above element?

A)



B)



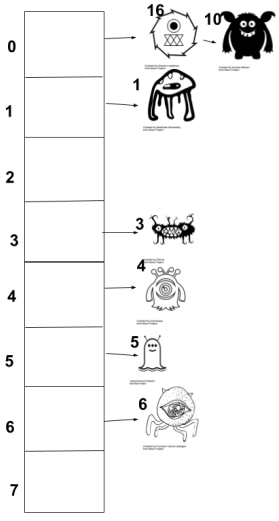


C)

( ) A
( ) B
( ) C

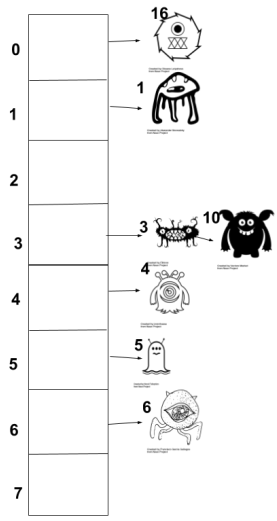
Part Two:
Now, after inserting that element, we want to insert a new element (shown below) **and** resize.

Which of the following correctly mirrors the new state?

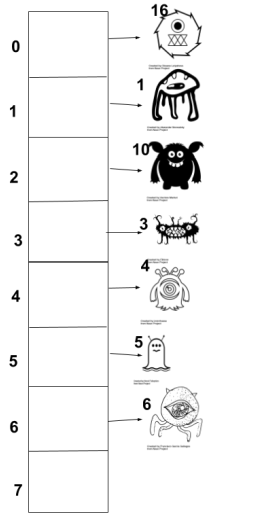


A)

B)



C)



( ) A
( ) B
( ) C

2 Unexpected Hashing

Suppose we have the `Lamp` class below:

```
1  class Lamp {
2      int brightness;
3
4      Lamp(int brightness) {
5          this.brightness = brightness;
6      }
7
8      @Override
9      public int hashCode() {
10         return brightness;
11     }
12
13     @Override
14     public boolean equals(Object o) {
15         return ((Lamp) o).brightness == brightness;
16     }
17 }
```

Assume the HashMap is implemented with external chaining. Assume the size of the internal array of the `HashMap` is 2 and doesn't resize. Determine the output of each print line below:

```
1  Lamp a = new Lamp(1);
2  Lamp b = new Lamp(2);
3
4  HashMap<Lamp, Integer> map = new HashMap<>();
5
6  map.put(b, 0);
7  map.put(a, 1);
8  map.put(a, 2);
9
10 System.out.println(map.get(a));  // print statement 1
11 System.out.println(map.get(b));  // print statement 2
12
13 map.put(b, 3);
14 a.brightness = 2;
15 map.put(b, 4);
16
17 System.out.println(map.get(a));  // print statement 3
18 System.out.println(map.get(b));  // print statement 4
19 System.out.println(map.get(new Lamp(1))); // print statement 5
```

Print Statement 1:

( ) 0
( ) 1
( ) 2
( ) 3
( ) 4
( ) null

Print Statement 2:

( ) 0
( ) 1
( ) 2
( ) 3
( ) 4
( ) null

Print Statement 3:

( ) 0
( ) 1
( ) 2
( ) 3
( ) 4
( ) null

Print Statement 4:

( ) 0
( ) 1
( ) 2
( ) 3
( ) 4
( ) null

Print Statement 5:

( ) 0
( ) 1
( ) 2
( ) 3
( ) 4
( ) null

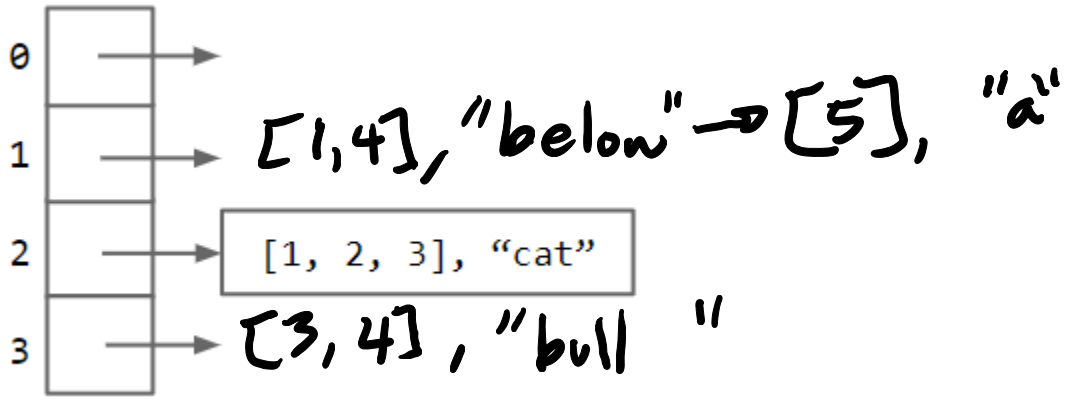
e) **(3 points).** Draw a valid BST of minimum height containing the keys 1, 2, 3, 7, 8 9, 5.

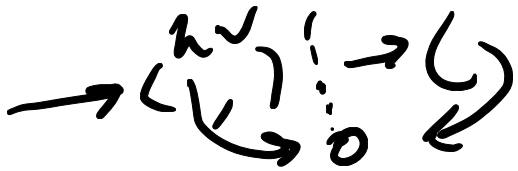
f) (6 points). Under what conditions is a **complete** BST containing N items **unique**? By unique we mean the BST is the only complete BST that contains exactly those N items. By complete we mean the same idea that was required for a tree to be considered a heap (not repeated here). Reminder: We never allow duplicates in a BST.

2. Hash Tables.

a) **(5 points).** Draw the hash table that is created by the following code. Assume that `XList` is a list of integers, and the hash code of an `XList` is the sum of the digits in the list. Assume that `XLists` are considered equal only if they have the same length and the same values in the same order. Assume that `FourBucketHashMaps` use external chaining and that new items are added to the end of each bucket. Assume `FourBucketHashMaps` always have four buckets and never resize. The result of the first `put` is provided for you. Represent lists with square bracket notation as in the example given.

```
FourBucketHashMap<XList, String> fbhm = new FourBucketHashMap<>();
fbhm.put(XList.of(1, 2, 3), “cat”);
fbhm.put(XList.of(1, 4), “riding”);
fbhm.put(XList.of(5), “a”);
fbhm.put(XList.of(3, 4), “bull”);
fbhm.put(XList.of(1, 4), “below”);
```





b) **(4.5 points).** Next to the calls to `get`, write the return value of the `get` call. Assume that `get` returns `null` if the item cannot be found.

```
FourBucketHashMap<XList, String> fbhm = new FourBucketHashMap<>();
XList firstList = XList.of(1, 2, 3);
fbhm.put(firstList, "cat");
fbhm.get(XList.of(1, 2, 3));          ____cat____
firstList.addLast(0); // list is now [1, 2, 3, 0]
fbhm.get(firstList);                  ____cat____
fbhm.get(XList.of(1, 2, 3));          ____null___
```

no change to hc

↳ right hc, wrong =

c) **(10.5 points).** Next to the calls to `get`, write the return value(s) of the `get` call. Assume that `get` returns `null` if the item cannot be found.

```
FourBucketHashMap<XList, String> fbhm = new FourBucketHashMap<>();
XList firstList = XList.of(1, 2, 3);
fbhm.put(firstList, "cat");
firstList.addLast(1); // list is now [1, 2, 3, 1]
fbhm.get(firstList);                  ____null___
fbhm.get(XList.of(1, 2, 3));          ____null___
fbhm.get(XList.of(1, 2, 3, 1));       ____null___
fbhm.get(XList.of(3, 4));             ____null___
fbhm.put(firstList, "dog");
fbhm.get(firstList);                  ____dog____
fbhm.get(XList.of(1, 2, 3));          ____null___
fbhm.get(XList.of(1, 2, 3, 1));       ____dog____
```

lost access

0
1
2 → fl, cat
3 → fl, dog

d) **(4 points).** What are the best and worst case `get` and `put` runtimes for `FourBucketHashMap` as a function of N, the number of items in the HashMap? Don't assume anything about the distribution of keys.

add

- `.get` best case: _Θ__1__
- `.get` worst case: _Θ__N__
- `.put` best case: _Θ__1__
- `.put` worst case _Θ__N__

worst

e) **(4 points).** If we modify `FourBucketHashMap` so that it triples the number of buckets when the load factor exceeds 0.7 instead of always having four buckets, what are the best and worst case runtimes in terms of N? Don't assume anything about the distribution of keys.

- `.get` best case: _Θ__1__
- `.get` worst case: _Θ__N__
- `.put` best case: _Θ__1__
- `.put` worst case _Θ__N__

As noted on the front page, throughout the exam you should assume that a single resize operation on any hash map takes linear time.