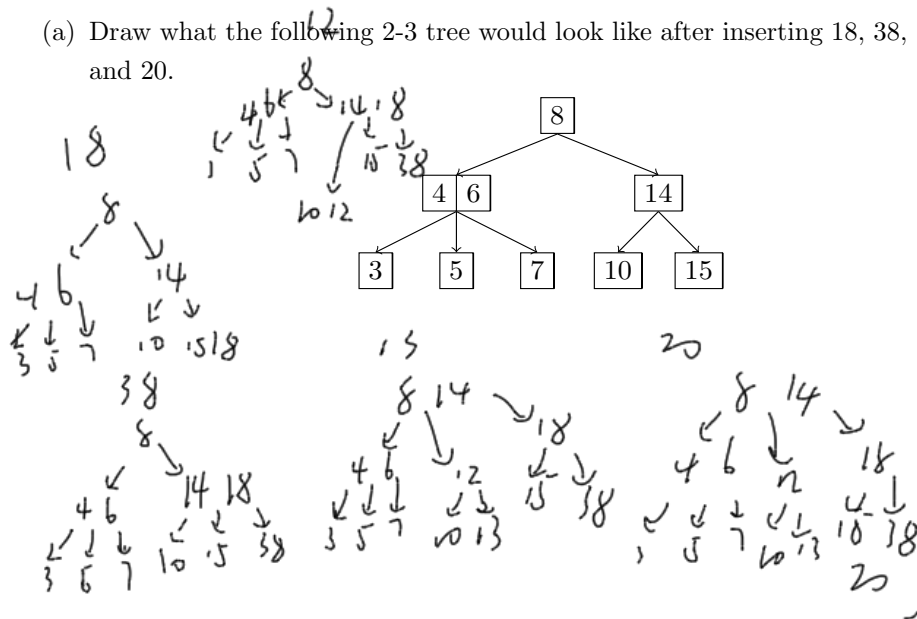
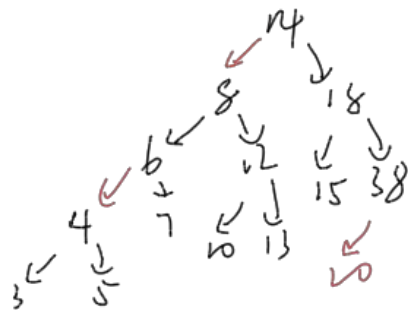


1 2-3 Trees and LLRB's

- (a) Draw what the following 2-3 tree would look like after inserting 18, 38, 12, 13, and 20.



- (b) Now, convert the resulting 2-3 tree to a left-leaning red-black tree.



- (c) If a 2-3 tree has depth H (that is, there are H number of edges in the path from leaf to the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

$$2H+2$$

2 Hashing

- (a) Here are three potential implementations of the `Integer`'s `hashCode()` function. Categorize each as either a valid or an invalid hash function. If it is invalid, explain why. If it is valid, point out a flaw or disadvantage. For the 2nd implementation, note that `intValue()` will return that `Integer`'s number value as an `int`.

```
public int hashCode() {
    return -1;
}
```

✓
100%冲突

```
public int hashCode() {
    return intValue() * intValue();
}
```

✓
正负数冲突

```
public int hashCode() {
    return super.hashCode();
}
```

X
super's object 好生之叉
返回地址编号 Integer x = Integer(5)
Integer y = Integer(5)
对 x y 地址不同
hashCode 不同
但地址编号相同
因为相等 (equals)

- (b) For each of the following questions, answer **Always**, **Sometimes**, or **Never**.

1. If you were able to modify a key that has been inserted into a `HashMap` would you be able to retrieve that entry again? For example, let us suppose you were mapping ID numbers to student names, and you did a `put(303, "Anjali")` operation. Now, let us suppose we somehow went to that item in our `HashMap` and manually changed the key to be 304. If we later do `get(304)`, will we be able to find and return "Anjali"? Explain.

Sometimes.
只修改键, 不会修改映射关系

2. When you modify a value that has been inserted into a `HashMap` will you be able to retrieve that entry again? For example, in the above scenario, suppose we first inserted `put(303, "Anjali")` and then changed that item's value from "Anjali" to "Ajay". If we later do `get(303)`, will we be able to find and return "Ajay"? Explain.

All ways.

3 A Side of Hashbrowns

We want to map food items to their yumminess. We want to be able to find this information in constant time, so we've decided to use java's built-in `HashMap` class! Here, the key is an `String` representing the food item and the value is an `int` yumminess rating.

For simplicity, let's say that here a `String`'s hashCode is the first letter's position in the alphabet (`A = 0, B = 1... Z = 25`). For example, the `String` "hashbrown" starts with "h", and "h" is 7th letter in the alphabet (0 indexed), so the hashCode would be 7. Note that in reality, a `String` has a much more complicated `hashCode()` implementation.

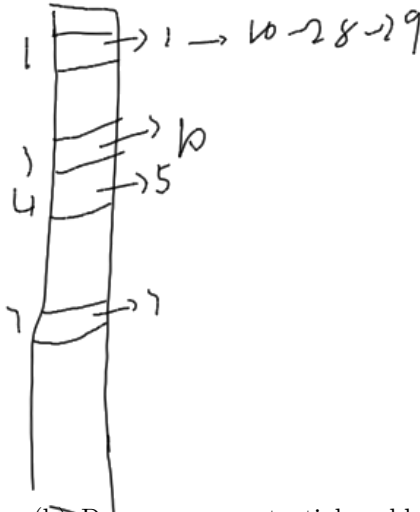
Our `HashMap` will compute the index as the key's hashCode value modulo the number of buckets in our `HashMap`. Assume the initial size is 4 buckets, and we double the size our `HashMap` as soon as the load factor reaches 3/4.

(a) Draw what the `HashMap` would look like after the following operations.

```

1  HashMap<Integer, String> hm = new HashMap<>();
2  hm.put("Hashbrowns", 7);
3  hm.put("Dim sum", 10);
4  hm.put("Escargot", 5);
5  hm.put("Brown bananas", 1);
6  hm.put("Burritos", 10);
7  hm.put("Buffalo wings", 8);
8  hm.put("Banh mi", 9);

```



冲突

(b) Do you see a potential problem here with the behavior of our hashmap? How could we solve this?