Q1 Modeling Recursive Code

2 Points

For (a) and (b), select the recurrence representing the recursive runtimes of each method in terms of `n`. Assume that any `+, -, *, /, %` operations and `System.out.println()` calls take constant runtime. Don't worry about finding the exact constants for the non-recursive term. For example, if the running time is $T(n) = 4T(n/3) + 25n$, you need to get the 4 and the 3 right but you don't have to worry about getting the 25 right.

Q1.1 From Code to Reccurence

1 Point

Write the recurrence of **the recursive case** (not the base case) for `mystery(n, 1)`, where the initial value of the parameter `step = 1`

```
/*
* A mystery method
*/
public void mystery(int n, int step) {
    if (n <= step) {
        return;
    }
    for (int i = 0; i < n; i += step) {
        int a = i + n / 2;
        System.out.print(a * a + " ");
    }
    System.out.println();
    mystery(n, step * 2);
    mystery(n, step * 2);
}
```

$T(n) = 3T(n/2) + n$

$T(n) = 2T(n/3) + 25$

$T(n) = 2T(n/2) + n$

$T(n) = 3T(n/3) + 25$

Save Answer

Q1.2 From Recurrence to Runtime
1 Point

Suppose we have a recurrence of the recursive case of a function defined as $T(N) = T(N/2) + N$. Use the Master Theorem as defined in lecture to find the Big-Theta runtime of the function.

- $\Theta(1)$
- $\Theta(logN)$
- $\Theta(N)$
- $\Theta(NlogN)$
- $\Theta(N^2)$
- $\Theta(N^2logN)$
- $\Theta(N^3)$
- $\Theta(2^N)$

Save Answer

Q2 Recurrence Relations with Case Analysis

3 Points

Consider the following method. Assume `k(N)` runs in constant time and returns a `bool`.

```
static void g0(int N) {
    if (N == 0)
        return;
    g0(N / 2);
    if (k(N))
        g0(N / 2);
}
```

Q2.1 Best-Case Situation

1 Point

In the best-case asymptotic runtime analysis for `g0`, `k(N)` always returns false.

True

False

Save Answer

Q2.2 Best-Case Asymptotic Runtime
1 Point

What is the best-case order of growth for the runtime of `g0` with respect to N?

Hint: This is a recurrence problem. Find the recurrence and match it to a runtime using methods discussed in lecture.

- $\Theta(1)$
- $\Theta(\log N)$
- $\Theta(N)$
- $\Theta(N \log N)$
- $\Theta(N^2)$
- $\Theta(2^N)$

Save Answer

Q2.3 Worst-Case Asymptotic Runtime
1 Point

What is the worst-case order of growth for the runtime of `g0` with respect to N?

Hint: This is a recurrence problem. Find the recurrence and match it to a runtime using methods discussed in lecture.

- $\Theta(1)$
- $\Theta(\log N)$
- $\Theta(N)$
- $\Theta(N \log N)$
- $\Theta(N^2)$
- $\Theta(2^N)$

Save Answer

Q3 Hash Functions Galore

2 Points

Q3.1

1 Point

Suppose we have a `Dog` class where each `Dog` has a `name` and a `weight`. Suppose we are trying to create the `hashCode` for our `Dog` class. Which of the two `hashCodes` below would guarantee to result in the two equivalent `Dog` objects, `d1` and `d2`, being hashed to the same bucket in our hash table?

```
Dog d1 = new Dog("Zeus", 5);
Dog d2 = new Dog("Zeus", 5);

public int hashCode1() {
    return (int)(Math.random() * 20) + 1;
}

public int hashCode2() {
    return (int)this.name.length();
}
```

hashCode1()

hashCode2()

Save Answer

Q3.2
1 Point

Suppose we want to write our own `hashCode` for the `Dog` class. Which of the following `hashCodes` will result in the most collisions overtime?

```
public int hashCode1() {
    return 17;
}
```

```
public int hashCode2() {
    return this.weight;
}
```

```
public int hashCode3() {
    return (int)this.name.length();
}
```

```
public int hashCode4() {
    int sumOfASCII = 0;
    for (int i = 0; i < this.name.length(); i++) {
      int asciiValue = this.name.charAt(i);
      sumOfASCII += asciiValue;
    }
    return sumOfASCII;
}
```

- hashCode1()
- hashCode2()
- hashCode3()
- hashCode4()

Save Answer

Q4 Hash Table Insertion

4 Points

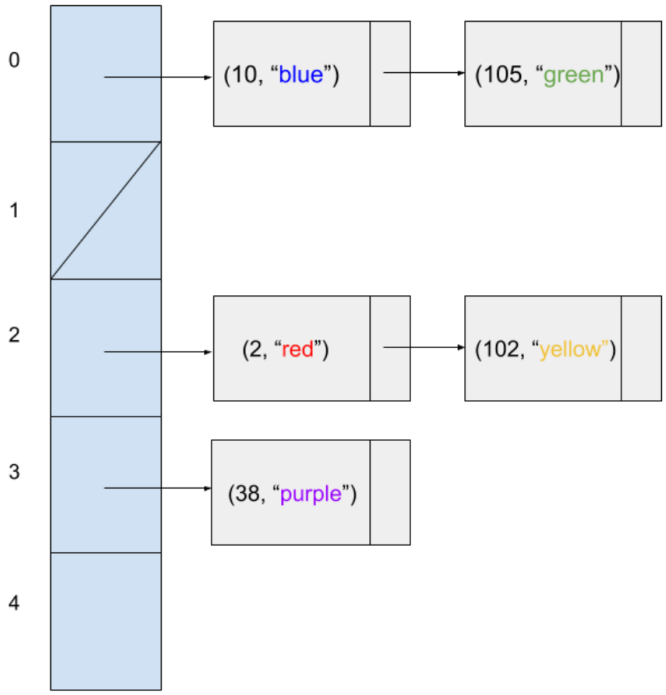
Consider the following code snippet.

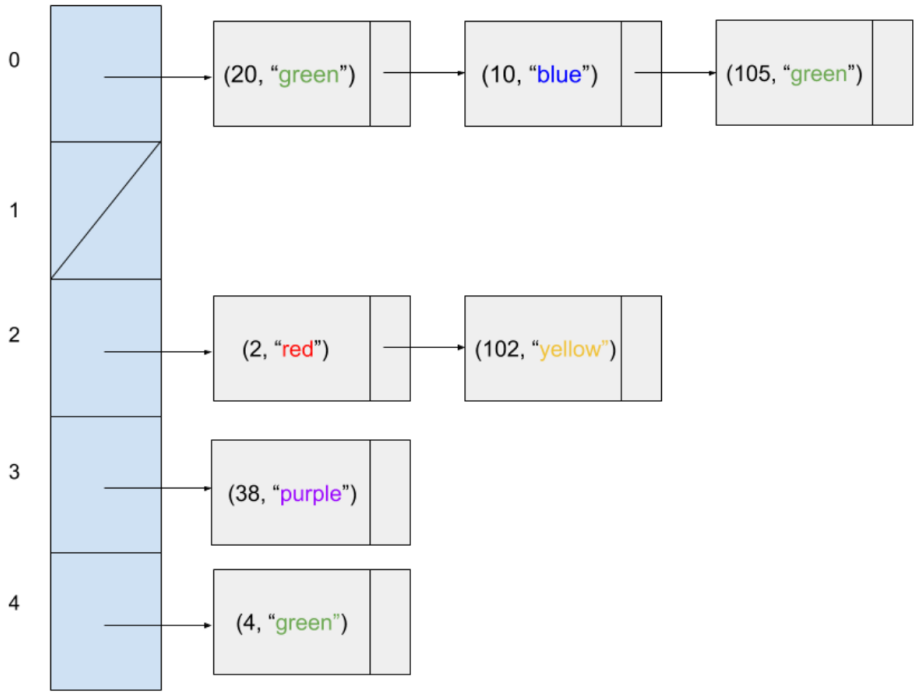
```
Map<Integer, String> map = new HashMap<>();
map.put(20,  "green");
map.put(2,   "red");
map.put(10,  "blue");
map.put(4,   "green");
map.put(105, "green");
map.put(102, "yellow");
map.put(38,  "purple");
```

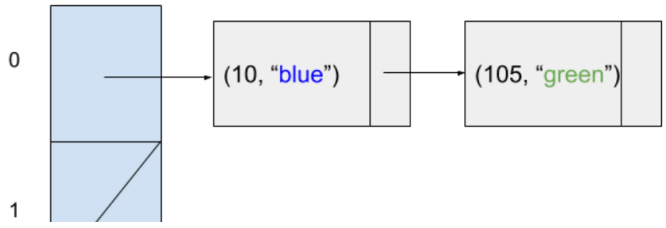
Q4.1
2 Points

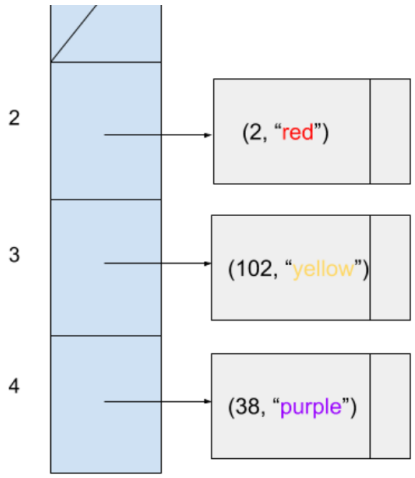
Assuming the `HashMap` does not resize and its underlying array is fixed at 5 buckets, which of the following diagrams might represent the state of the hash table at the end of the code snippet?

For this question, assume that the `hashCode` simply returns the `HashMap.key`'s `int` value to then be reduced by modulo by the table size of 5 with no resizing (i.e. our `hashCode` operation is `HashMap.key % 5`).


0
(10, “blue”)
(105, “green”)
1
2
(2, “red”)
(102, “yellow”)
3
(38, “purple”)
4


0
(20, “green”)
(10, “blue”)
(105, “green”)
1
2
(2, “red”)
(102, “yellow”)
3
(38, “purple”)
4
(4, “green”)


0
(10, “blue”)
(105, “green”)
1

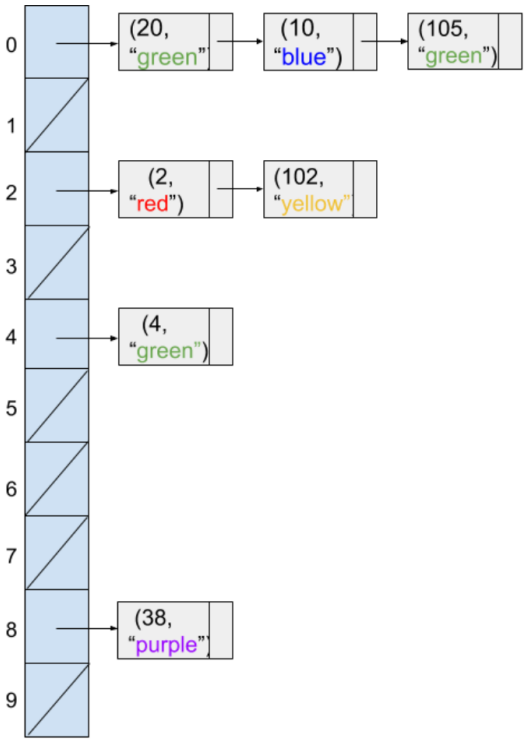


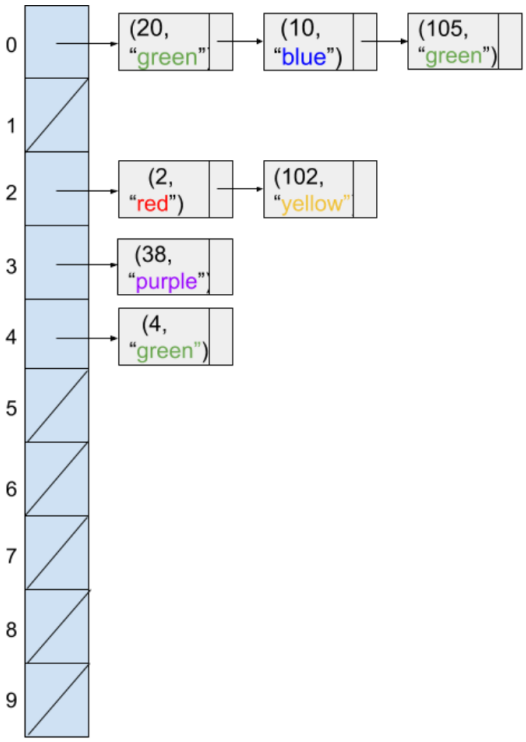
Save Answer

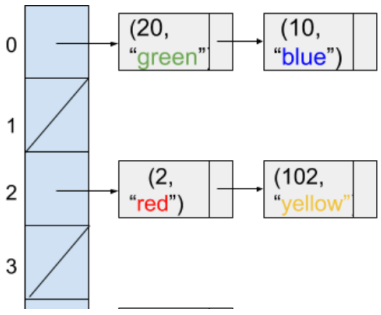
Q4.2
2 Points

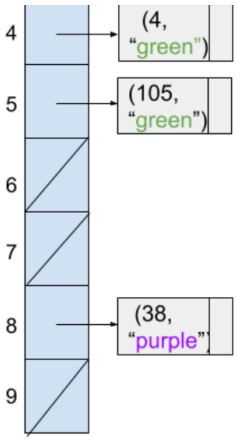
Assuming the `HashMap` resizes its underlying array when the number of items equals the number of buckets by doubling the number of buckets (from 5 buckets to 10 buckets), which of the following diagrams might represent the state of the hash table at the end of the code snippet?

For this question, assume that the `hashCode` simply returns the `HashMap.key`'s `int` value to then be reduced by modulo by the table size of 10 with no resizing (i.e. our `hashCode` operation is `HashMap.key % 10`).


0
1
2
3
4
5
6
7
8
9
(20, "green")
(10, "blue")
(105, "green")
(2, "red")
(102, "yellow")
(4, "green")
(38, "purple")


0
1
2
3
4
5
6
7
8
9
(20, "green")
(10, "blue")
(105, "green")
(2, "red")
(102, "yellow")
(38, "purple")
(4, "green")


0
1
2
3
(20, "green")
(10, "blue")
(2, "red")
(102, "yellow")



Save Answer

Q5 Hashing Strategies

2 Points

For the following problems, assume that:

- The `hashCode` of a FourBucketHashMap is the length of the first String of the object (which is to be treated as a regular `key` in a regular `HashMap`).
- FourBucketHashMap uses separate chaining and the new items are added to the back of each bucket.
- FourBucketHashMap always has four buckets and never resizes.

Consider the following code:

```
FourBucketHashMap<String, String> fbhm = new FourBucketHashMap<>();
fbhm.put("animal", "dog");
// Part i
fbhm.put("animal", "cat");
// Part ii
```

Q5.1

1 Point

At Part i, what will be returned from the following statement?

```
fbhm.get("animal");
```

null

"cat"

"dog"

Save Answer

Q5.2
1 Point

At Part ii, what will be returned from the following statement?

```
fbhm.get("animal");
```

null

"cat"

"dog"

Save Answer

Q6 Your turn to Hash!

5 Points

Assume you are hashing a set (unknown length) of randomly generated `Strings`, into a `HashTable` with a a size of 50 (this means your `HashTable` has 50 buckets). You can also assume that `Strings` are iterable and `Characters` utilize the Java native hash function.

Examine the two `hashCode()` implementations below. In general, which `hashCode()` option will result in a **greater** number of collisions for all `Strings`? Justify your answer in 1-2 sentences describing how each hashCode may generate a range of possible hashCode values.

Assume that when we call `this` in our hash functions, we are referring to each `String` object we are hashing.

Option 1:

This code utilizes Java's implementation of `hashCode()` for Characters which returns the unique int value associated with each character based on its assigned ASCII value. Ex: 'a' returns 97, 'A' returns 65.

```
public int hashCode1() {
        Iterator<Character> iterator = this.iterator();
        int result = 0;
        int i = 0;
        while (iterator.hasNext()) {
            result += iterator.next().hashCode();
            i++;
        }
        return result;
}
```

Option 2:

This code utilizes Java's implementation of `hashCode()` for Strings which is the following (value is an array of the characters within the array):

```
//Java's String hashCode implementation
public static int hashCode(byte[] value) {
    int h = 0;
    int length = value.length;
    for (int i = 0; i < length; i++) {
        h = 31 * h + getChar(value, i);
    }
```

```
    return h;
}
```

```
public int hashCode2() {
        return this.hashCode().
}
```

Save Answer

Save All Answers

Submit & View Submission >