Review Study Questions for MT3

Questions for your own practice and learning

```
int key = 0;
for (String charString : input.split("")) {
    key = charString.charAt(0) + (33 * key);
}
return key % hashtable.length;
```

1. Hashing

(a) Create a hashing function that uniformly distributes the strings "python", "java", "c", "c++", "ocaml" into 5 buckets.

(b) For the previous question, what do you know about your hash table if there are less than 5 buckets?

it is a shit hash table. Jk Pigeonhole Principle Moment. Will be collisions!

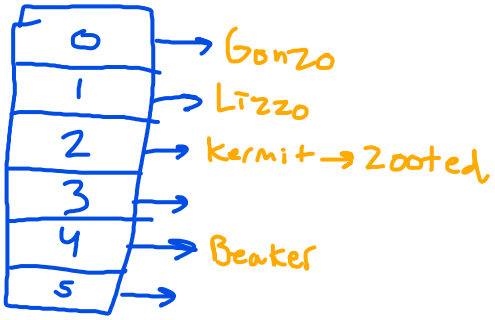
(c) What happens if the range of the hashing function is larger than the number of buckets? Can this be fixed easily? Stinky will error.

Mod keyReturn by amount of buckets!

(d) Assume we have a hash table of size 5 and a hashing function that maps the following strings as such:
"Kermit" : 2
"Beaker" : 4
"Lizzo" : 1
"Gonzo" : 0
"Zoot" : 2
Construct the hash-table that would result after adding all of the strings using open addressing.

0	1	2	3	4	5
Gonzo	Lizzo	Kermit	Zooted	Beaker	

(e) Draw the hash-table that would result after adding all of the strings using chaining.



Pre: root left right
in: Left root right
Post: Left right root

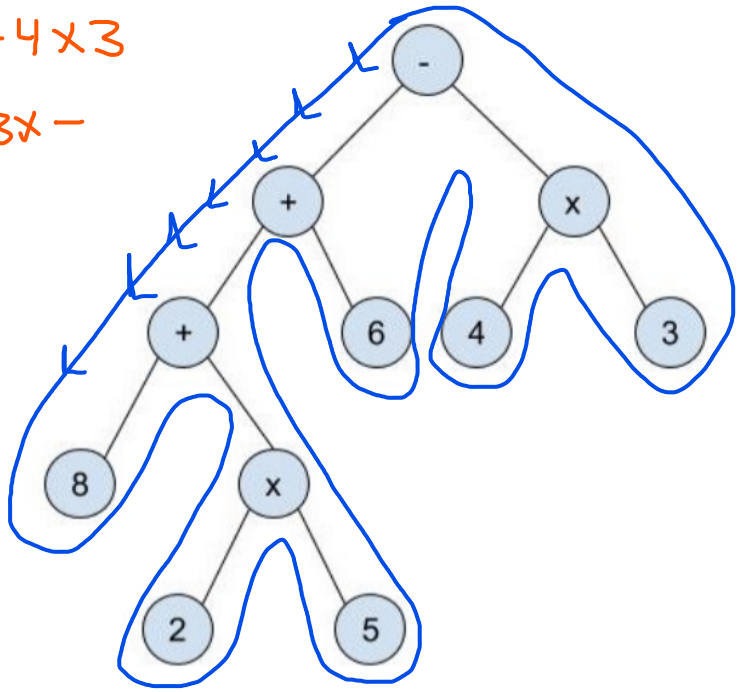
2. Binary Trees

(a) Write a pre-order, in-order, and post-order node traversal of the following binary tree

Preorder: −++8x256x43

inorder: 8+2x5+6−4x3

Postorder: 825x+6+43x−



(b) Complete the method equals() as part of the BTNode class below. equals() should return true if the two trees have identical structure with the data in the same locations.

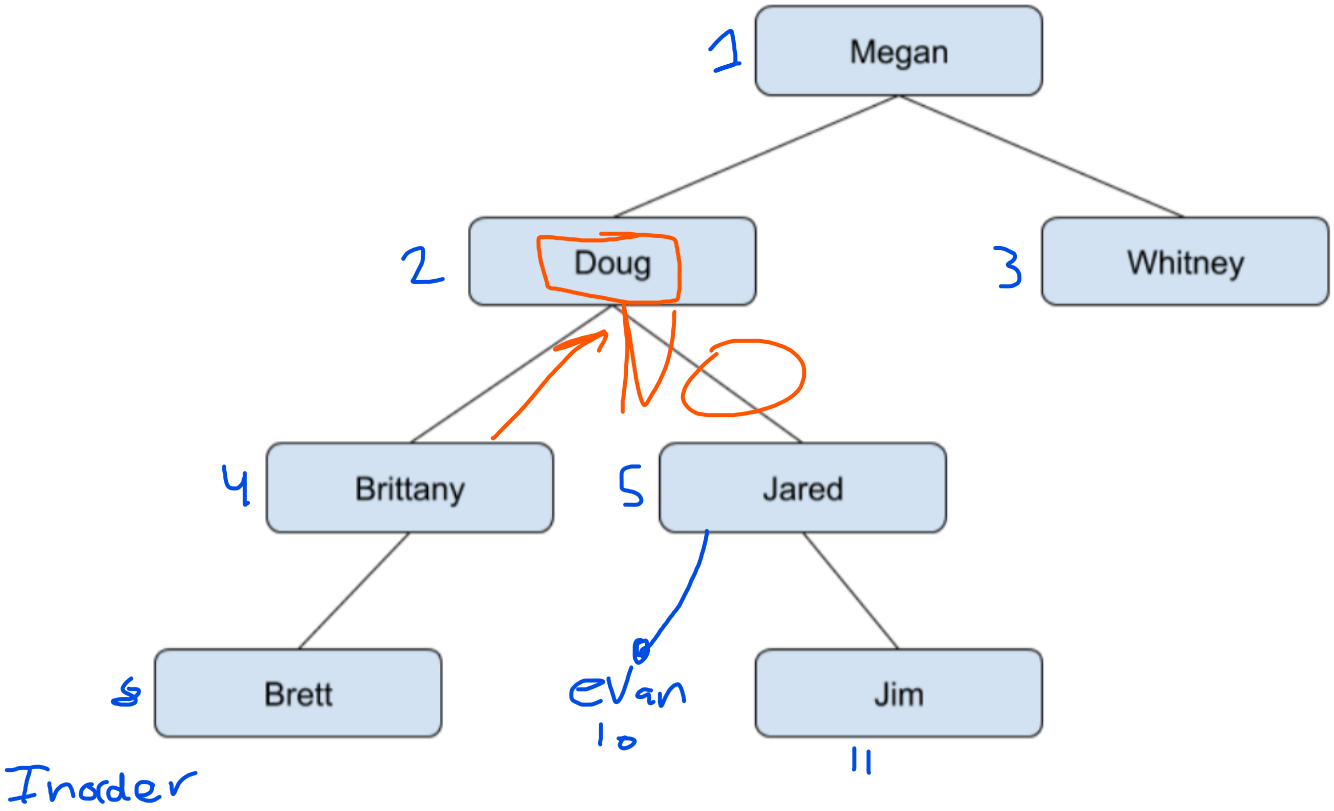
```
public class BTNode<T extends Comparable<T>>{
    private T data;
    private BTNode<T> left;
    private BTNode<T> right;

    public BTNode<T> getLeft() { return left; }
    public BTNode<T> getRight() { return right; }
    public T getData() { return data; }

    public void setLeft(BTNode<T> l) { left = l; }
    public void setRight(BTNode<T> r) { right = r; }
    public void setData(T d) { data = d; }

    public boolean equals(BTNode<T> o){} // your code here
```

3. Binary Search Trees



(a) List the series of nodes visited for binary search of the target name "Chad".

Megan → Doug → Brittany

(b) Repeat (a) for the target name "Whitney".

Megan → whitney

(c) Write an array representation of the tree after inserting the name "Evan".

Megan Doug whitney Brittany Jared null null Brett null evan Jim

(d) Using the result of (c), write an array representation of the tree after removing the name "Doug".

(e) What's the worst-case time complexity of a search on this binary search tree? In general?

O(n) worst O(log n) general

4. Binary Search Algorithm

(a) In the binary search algorithm, what are the four cases you have to consider at each iteration and what do you have to do in each case?

Case 1: found item → return item

Case 2: item in a[0] → a[middle - 1] → call function recursively

Case 3: item in a[middle + 1] → a[end]

Case 4: not in array / tree →

(b) What is the worst case time complexity of the binary search algorithm if implemented on a sorted array compared to the worst case time complexity if implemented on a binary search tree? Explain.

5. Generic Stacks and Queues

(a) Assuming that you have completed implementations of Stack and Queue with the partial classes below, complete the instance method reverse() of the Queue class to reverse the order of the elements in the Queue.

```
public class Queue<T> {
    public void enqueue(T item){...}
    public T dequeue(){...}
    public boolean isEmpty(){...}
    public void reverse(){...} //You need to implement this method
}

public class Stack<T>{
    public void push(T item){...}
    public T pop(){...}
    public boolean isEmpty(){...}
}

public void reverse() { //Your code here



}
```

(b) What is wrong with the following new method associated with Queue? There may be more than one error.

```
public T returnMaxElement() {
  Queue<T> tempQueue = new Queue<T>();
  T max = null;
  while(!this.isEmpty()) {
    T element = this.dequeue();
    if(max < element || max == null) {
      max = element;
    }
  }
  return max;
}
```

6. Merge Sort

(a) In what cases should Merge Sort be used? In what cases is it not efficient? Explain.

(b) The following is an implementation of merge and mergeSort.

```
public static void merge(int[] a, int start, int mid, int end, int[] temp) {
    int ptr1 = start;
    int ptr2 = mid + 1;
    int resPtr = start;

    while (ptr1 <= mid && ptr2 <= end) {

        System.out.println("" + a[ptr1] + " " + a[ptr2]); // Note this line

        if (a[ptr1] <= a[ptr2]){
          temp[resPtr++] = a[ptr1++];
        }
        else temp[resPtr++] = a[ptr2++];
    }

    if (ptr1 <= mid)
      for (int i = ptr1; i <= mid; i++)
        temp[resPtr++] = a[i];
    else
      for (int i = ptr2; i <= end; i++)
        temp[resPtr++] = a[i];
    System.arraycopy(temp, start, a, start, end - start + 1);
} // merge
```

```
public static void mergeSort(int[] a, int start, int end, int[] temp) {
    if (start < end) {
        int mid = (start + end) / 2;
        mergeSort(a, start, mid, temp);
        mergeSort(a, mid + 1, end, temp);
        merge(a, start, mid, end, temp);
    }
} // mergeSort
```

Making sure to take note of the added print statement in merge, what would the following call in main print to the screen?

```
int[] a = {4, 3, 2, 1};
int[] temp = new int[a.length];
mergeSort(a, 0, a.length-1, temp);
```

7. Complexity

(a) What are the best, worst, and average case runtimes for finding the smallest element in an unsorted array? Explain.

Best: O(1) average: O(n) worst O(n)

(b) If you know that a binary tree is full, compare the complexity of finding the number of elements in the binary tree with finding the number of elements in a perfect binary tree. Explain.

Full: O(log n)

(c) Hash tables are normally very efficient - often O(1) with a good hash function. However, hash tables are not always the best choice. When would it be better to choose a different data representation? Explain.

when data need to be in order

(d) In what cases would an array representation of a binary tree be more efficient than a node representation? Explain.

few insertions & many searches