CS 1332 Exam 3B

TOTAL POINTS

97 / 100

QUESTION 1

1 Hash Map - Coding 15 / 15

✓ + 3 pts Calls regrowBackingTable()

✓ + 1 pts Correct resize logic (correct math/not integer division, strict inequality, resizes to 2 * table.length)

✓ + 2 pts Computes the hash correctly (abs(hash % length) or abs(hash) % length)

+ 2 pts Loops through the external chain with correct bounds

✓ + 2 pts Replaces an existing key's value

✓ + 4 pts Adds to the back of the chain (unless they check for duplicates correctly and still add to the back when one exists)

✓ + 1 pts Size is incremented if and only if a new entry is added

+ 0 pts Incorrect / No answer

- 1 pts Syntax error

- 2 pts Efficiency (unrelated to other rubric items)

+ 2 Point adjustment

Your bounds do not work where there is only 1 element in the list. You will get a null pointer exception; +2: in-class quiz

QUESTION 2

2-4 Trees - Diagram 10 pts

2.1 Adding 4 / 4

✓ + 1 pts 19 is added and satisfies the order property

✓ + 2 pts 15 is promoted to its parent

✓ + 1 pts 12 is promoted to its parent

+ 2 pts Promotes both 18 and 20 instead of 15 and 12

+ 0 pts Incorrect / No answer

- 2 pts Splits incorrectly after promotion

2.2 Removing 6 / 6

✓ + 1 pts 13 is not in the tree and the tree follows the order property

✓ + 1 pts The resulting tree follows the shape property

✓ + 2 pts 10 is at the root (by itself)

✓ + 2 pts Correctly does transfer

+ 0 pts Incorrect / No answer

QUESTION 3

Radix Sort - Diagram 10 pts

3.1 Iteration 1 buckets 3 / 3

✓ + 1 pts All of the data (7 elements) are in the buckets

✓ + 1 pts The buckets contain the correct data (0 -> 1090, 2-> 1332 and 1502, 3 -> 1153, 6 -> 4026 and 3426, 8 -> 1098)

✓ + 1 pts The data are in the correct order in the buckets (1332 above 1502 AND 4026 above 3426)

+ 0 pts Incorrect / No answer

3.2 Iteration 1 array 2 / 2

✓ + 1 pts 1090, 1332 and 1502, 1153, 4026 and 3426, 1098

✓ + 1 pts 1332 is before 1502 and 4026 is before 3426

+ 0 pts Incorrect / No answer

3.3 Iteration 2 buckets 3 / 3

✓ + 1 pts All of the data (7) are in the buckets

✓ + 1 pts The data are in the correct buckets (0 -> 1502, 2 -> 4026 and 3426, 3 -> 1332, 5 -> 1153, 9 -> 1090 and 1098)

✓ + 1 pts The data are in the correct order in the buckets (4026 above 3426 and 1090 above 1098)

+ **0 pts** Incorrect / No answer

3.4 Iteration 2 array **2 / 2**

✓ **+ 1 pts** **The array contains 1502, 4026 and 3426, 1332, 1153, 1090 and 1098**

✓ **+ 1 pts** **4026 is immediately before 3426 and 1090 is immediately before 1098**

+ **0 pts** Incorrect / No answer

QUESTION 4

4 Boyer-Moore - Tracing **15 / 15**

✓ **+ 2 pts** **Always starts comparisons from the end of the pattern**

✓ **+ 3 pts** **Always stops comparing when there is a mismatch**

✓ **+ 1 pts** **Pattern shifts by 1 at least once**

✓ **+ 1 pts** **Pattern jumps past mismatch at least once**

✓ **+ 1 pts** **Pattern jumps past mismatch a second time**

✓ **+ 1 pts** **Mismatched character aligns with its last occurrence at least once**

✓ **+ 4 pts** **All mismatches and shifts are exactly right. First mismatch occurs at text[2], pattern shifts by 1. Second mismatch occurs at text[4], pattern shifts past it. Third mismatch occurs at text[8], pattern shifts past it. Fourth mismatch occurs at text[11], pattern aligns 'f' with mismatch.**

✓ **+ 2 pts** **Match from text[11] to text[14] is found on the last line**

+ **0 pts** Incorrect / No answer

QUESTION 5

Sorting Algorithms - Properties and Efficiency 10 pts

5.1 Stability/In-Place **3 / 4**

✓ **+ 1 pts** **LSD is stable and NOT in-place**

+ **1 pts** Merge is stable and NOT in-place

✓ **+ 1 pts** **QuickSort is NOT stable and in-place**

✓ **+ 1 pts** **Cocktail Shaker is stable and in-place**

+ **0 pts** Incorrect / No Answer

5.2 Average case Quick Select **2 / 2**

+ **0 pts** O(1)

+ **0 pts** O(log n)

✓ **+ 2 pts** **O(n)**

+ **0 pts** O(n log n)

+ **0 pts** O(n^2)

+ **0 pts** Incorrect / No Answer

5.3 Worst case KMP **2 / 2**

+ **0 pts** O(1)

+ **0 pts** O(m)

+ **0 pts** O(n)

✓ **+ 2 pts** **O(n + m)**

+ **0 pts** O(nm)

+ **0 pts** Incorrect / No Answer

5.4 Worst case Quick Sort **2 / 2**

+ **0 pts** O(1)

+ **0 pts** O(log n)

+ **0 pts** O(n)

+ **0 pts** O(n log n)

✓ **+ 2 pts** **O(n^2)**

+ **0 pts** Incorrect / No Answer

QUESTION 6

AVL - Fill in the Blank Coding 15 pts

6.1 Height/balance factor update **5 / 5**

✓ **+ 0.5 pts** **Uses 2 or more lines**

✓ **+ 0.5 pts** **Updates height**

✓ **+ 2 pts** **Updates height correctly (Math.max(leftH, rightH) + 1)**

✓ **+ 0.5 pts** **Updates bf**

✓ **+ 1.5 pts** **Updates bf correctly (leftH - rightH)**

- **1 pts** Syntax error

+ **0 pts** Incorrect / No Answer

- **1 pts** Inefficient

6.2 Rotation **5 / 5**

✓ **+ 1 pts** **Is exactly 2 lines**

✓ **+ 2 pts** **Sets node's left to B's right**

✓ **+ 2 pts** **Set's B's right to node**

- **1 pts** Syntax (getter/setter, naming, etc.)

- **1 pts** Create new node reference
- **1 pts** Modified tree data
+ **0 pts** Incorrect / No Answer

6.3 Update after rotation 5 / 5

✓ **+ 1 pts** **Is exactly 2 lines**
✓ **+ 1 pts** **Updates the height/bf of node**
✓ **+ 1 pts** **Updates the height/bf of B**
✓ **+ 2 pts** **Updates node before B**
+ **0 pts** Incorrect / No Answer
- **1 pts** Syntax Error

QUESTION 7

Quick Sort - Fill in the Blank Coding 15 pts

7.1 Base case 3 / 3

✓ **+ 1 pts** **Is 3 or more lines (braces count as lines)**
✓ **+ 1 pts** **Checks if left >= right or some equivalent**
✓ **+ 1 pts** **Breaks out of the method if the condition is true (calls return)**
+ **0 pts** Incorrect / No Answer
- **1.5 pts** Unsorts the array/affects normal behavior of the algorithm
- **0.5 pts** Syntax

7.2 Move i 3 / 3

✓ **+ 0.5 pts** **Has a while loop**
✓ **+ 0.5 pts** **Increments i**
✓ **+ 1 pts** **iterates while i <= j**
✓ **+ 1 pts** **Iterates while arr[i] <= pivot or arr[i] < pivot**
+ **0 pts** Incorrect / No Answer
- **0.5 pts** Affects operation of quickSort in an unrelated way
- **0.5 pts** syntax error

7.3 Move j 2 / 3

✓ **+ 0.5 pts** **Has a while loop**
+ **0.5 pts** Decrements j
✓ **+ 1 pts** **Iterates while i <= j**
✓ **+ 1 pts** **Iterates while arr[j] >= pivot or arr[j] > pivot**
+ **0 pts** Incorrect / No Answer
- **0.5 pts** Affects operation of quickSort in an unrelated way
✓ **- 0.5 pts** **Syntax error**
arr[j] should be compared to pivot, not arr[pivot]

7.4 Swap pivot 3 / 3

✓ **+ 1.5 pts** **The element at index j swaps with index left/pivot**
✓ **+ 1.5 pts** **The swap is done correctly**
+ **0 pts** Incorrect / No Answer
- **0.5 pts** Syntax error

7.5 Recursive call 2 / 3

✓ **+ 1 pts** **Calls quicksort twice**
✓ **+ 1 pts** **The bounds on one call are left and j - 1**
+ **1 pts** The bounds on the other call are j + 1 and right
- **1 pts** Syntax
+ **0 pts** Incorrect / No Answer

QUESTION 8

8 KMP - Failure Table Diagram 10 / 10

✓ **+ 1 pts** **f[0] = 0**
✓ **+ 0.5 pts** **f[1] = 0**
✓ **+ 0.5 pts** **f[2] = f[1] + 1 --- (1)**
✓ **+ 0.5 pts** **f[3] = 0**
✓ **+ 0.5 pts** **f[4] = f[3] + 1 --- (1)**
✓ **+ 1 pts** **f[5] = f[4] + 1 --- (2)**
✓ **+ 1 pts** **f[6] = f[5] + 1 --- (3)**
✓ **+ 4 pts** **f[7] = f[6] - 1 --- (2)**
✓ **+ 1 pts** **f[8] = f[7] + 1 --- (3)**
+ **0 pts** Incorrect / No Answer

CS 1332 Exam 3B
Spring Semester 2018 - April 11, 2018

Name (print clearly including your first and last name): _____

Signature: _____

GT account username (msmith3, etc): _____

GT account number (903000000, etc): _____

- You must have your BuzzCard or other form of identification on the table in front of you during the exam. When you turn in your exam, you will have to show your ID to the TAs before we will accept your exam. It is your responsibility to have your ID prior to beginning the exam.
- You are not allowed to leave the exam room and return. If you leave the room for any reason, then you must turn in your exam as complete.
- Signing and/or taking this exam signifies you are aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct.
- Notes, books, calculators, phones, laptops, smart watches, headphones, etc. are not allowed.
- Extra paper is not allowed. If you have exhausted all space on this test, talk with your instructor. There are extra blank pages in the exam for extra space.
- Pens/pencils and erasers are allowed. Do not share.
- All code must be in Java.
- Efficiency matters. For example, if you code something that uses O(n) time or worse when there is an obvious way to do it in O(1) time, your solution may lose credit. If your code traverses the data 5 times when once would be sufficient, then this also is considered poor efficiency even though both are O(n).
- Style standards such as (but not limited to) use of good variable names and proper indentation is always required. (Don't fret too much if your paper gets messy, use arrows or whatever it takes to make your answer clear when necessary.)
- Comments are not required unless a question explicitly asks for them.

HEYYYYYYY if you're looking at this exam and you're in the class, I uploaded a word doc of what questions I rmbr from the Final. GL

This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.

1) Hash Map - Coding [15 points]

Below you will find some starter code for the HashMap class. You are responsible for implementing the `add()` method. You may **NOT** import any libraries, create any new instance variables (local variables are fine), or write any helper methods. Your code must be as efficient as possible. Remember that the instance variables of private inner classes can be accessed and modified directly without getter or setter methods. Note that you are given a regrow method to use. You do not need to implement your own.

```
public class HashMap<K, V> {
    private class MapEntry<K, V> {
        K key;
        V value;
        MapEntry<K, V> next;
        // Omitted is a constructor for (K key, V value, MapEntry<K, V> next)
        // as well as (K key, V value)
    }

    private MapEntry<K, V>[] table;
    private int size;

    private void regrowBackingTable(int length) {
        // implementation omitted. Table will be resized to length and the
data
        // replaced correctly into the table
    }

    /**
     * Add a MapEntry to the backing table with the given
     * key and value.
     *
     * If there is a collision, use the external chaining
     * resolution strategy. Add new entries to the end
     * of existing chains. The new key may already exist in
     * the Map; replace the value in the existing entry with
     * the new value. If the load factor is strictly greater than
     * 0.75, regrow the table to double the previous length.
     *
     * @param key the new key to be added, will never be null
     * @param value the new value to be added, will never be null
     */
    public void add(K key, V value) {
        if((double)(size)/(double)(table.length) > 0.75){
            regrowBackingTable((table.length * 2));
        }
        //cthd next pg.
    //}
```

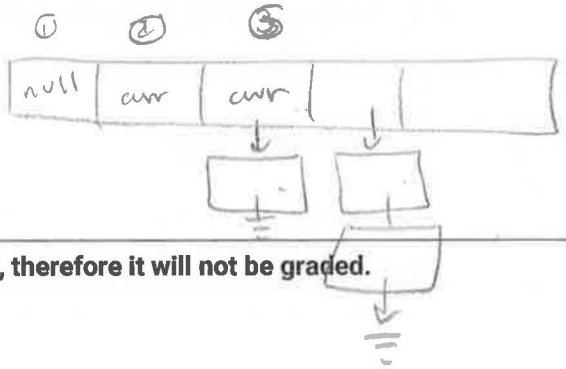
cthd next pg.

```
//}

int hash = Math.abs(key.hashCode() % table.length);
if (table[hash] == null) {
    MapEntry<K,V> ent = new MapEntry(key, value);
    table[hash] = ent;
    size++;
    return;
} else {
    MapEntry<K,V> curr = table[hash];

    do {
        if ((curr.key).equals(key)) {
            curr.value = value;
            return;
        } else {
            curr = curr.next;
        }
    }while (curr.next != null) {
    MapEntry<K,V> ent = new MapEntry(key, value);
    curr.next = ent;
    size++;
    return;
} //end else
} //end method

// question for HashCode
```

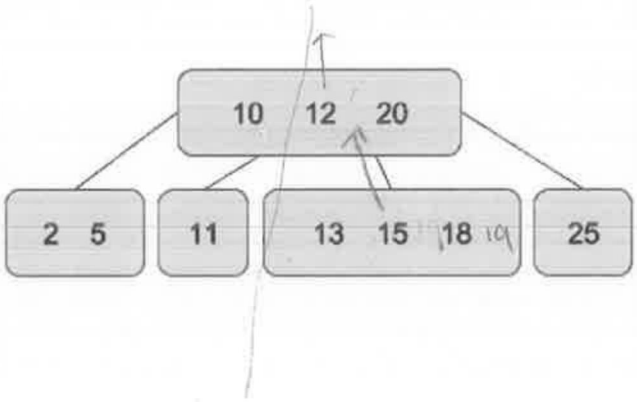


This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.

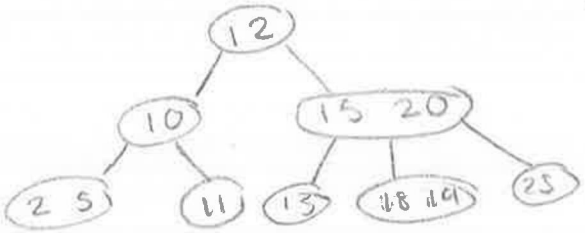
2) (2, 4) Trees - Diagram [10 points]

Given both of the following (2, 4) trees below, perform the specified add or remove operation and draw the new tree in the box to the right of the original tree. If you want to show multiple steps in a box, circle the final tree. If necessary for adding, use the **2nd** element. If necessary for removing, use the **predecessor**.

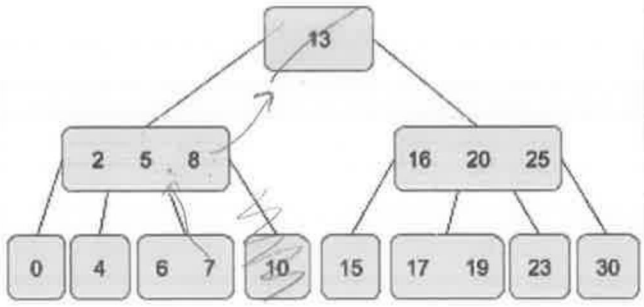
A)



Add 19 to part A:

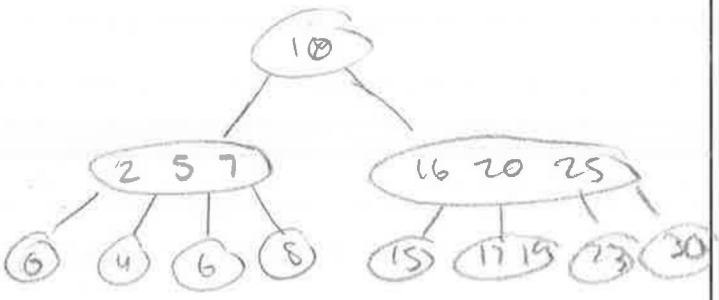


B)



10

Remove 13 from part B:



3) Radix Sort - Diagram [10 points]

Perform the first **two** iterations of LSD radix sort on the provided array. You must show the contents of each bucket during each iteration. When writing the contents of a bucket, write the items in the order you add them, from **top to bottom**. Assume the algorithm accepts only positive numbers (and uses ten buckets).

Initial array:

i	0	1	2	3	4	5	6
a[i]	4026	1153	1332	1502	3426	1090	1098

Iteration 1 buckets:

0	1	2	3	4	5	6	7	8	9
1090		1332 1502	1153			4026 3426		1098	

Array after iteration 1:

i	0	1	2	3	4	5	6
a[i]	1090	1332	1502	1153	4026	3426	1098

Iteration 2 buckets:

0	1	2	3	4	5	6	7	8	9
1502		4026 3426	1332		1153				1090 1098

Array after iteration 2:

i	0	1	2	3	4	5	6
a[i]	1502	4026	3426	1332	1153	1090	1098

4) Boyer-Moore - Tracing [15 points]

Given the following text, pattern, and last occurrence table, perform the **Boyer-Moore** algorithm to find the **first** matching substring in the text. Starting at the top of the table below, align the pattern with the text, then **circle each character in the pattern when they are compared with the text.** In the event of a mismatch, realign the pattern using the next row of the table.

- check the text char when mismatch
- start from BACK
- restart from BACK in comps
- if back 1, shift +1

Text: **"free the referee"**

Pattern: **"fere"**

Last Occurrence Table:

f	e	r	*
0	3	2	-1

GOOS +1

Matching:

f	r	e	e	-	t	h	e	-	r	e	f	e	r	e	e
f	e	r	e												
	f	e	r	e											
					f	e	r	e							
									f	e	r	e			
											f	e	r	e	

5) Sorting Algorithms - Properties and Efficiency [10 points]

For this first section below, place a **checkmark** in the box if the sorting algorithm satisfies the property shown. **Leave it blank** if it does not have that property. If no modification of the algorithm is specified, use the version that was taught in class.

Sorting Algorithm	Stable	In-Place
LSD Radix Sort	✓	
Merge Sort		
Quick Sort (as taught in class)		✓
Cocktail Shaker Sort	✓	✓

Merge = O(n log n).

Quick =

For the section below, determine the efficiency of each operation. Select the bubble corresponding to your choice by **filling in the circle** next to your choice. Make sure you choose the tightest Big-O upper bound possible for the operation.

A.) **Average** case time complexity of randomized Quick Select.

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ O(n^2)

B.) **Worst** case time complexity of the KMP string searching algorithm.

○ O(1) ○ O(~~log~~ m n) ○ O(n) ● O(~~n log~~ n + m) ○ O(~~n^2~~ nm)

C.) **Worst** case time complexity of Quick Sort.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ● O(n^2)

6) AVL - Fill in the Blank Coding [15 points]

The following code has a method that performs a single right rotation on a node in an AVL tree and another method to update heights and balance factors. However, some lines are left open for you to implement. There will be boxes for you to fill in the lines of code missing. The boxes contain italicized instructions that you *must* follow as well as requirements in bold. For the entire question, you **CANNOT** create a new variable name to reference a node. This means at **NO POINT** can you write `Node anything = ....` We will give you all of the references you need to correctly perform the rotation. You may use `Math.max(int a, int b)`. You have access to **NO** other methods within the AVL class.

```
public class AVL {

    private class AVLNode {
        int data, height, bf;
        AVLNode left, right;
    }

    private AVLNode root;
    private int size;

    /**
    * Updates the height and balance factor of a given node based on the
    * heights of the children
    * @param node the node that is updated
    */
    private void updateNode(AVLNode node) {
        int leftH = node.left != null ? node.left.height : -1;
        int rightH = node.right != null ? node.right.height : -1;
```

Update the height and balance factor on the node. ***[2 or more lines]***

```
node.height = (Math.max(leftH, rightH) + 1);
node.bf = leftH - rightH;
```

```
    }

    // Question continued on the next page
```

```
/**
* A right rotation is a rotation of the following form:
*      C
*     /
*    B      =>       B
*   /              /   \
*  A              A     C
* Note, this is just an example and is not exhaustive.
* The node passed in will definitely need a right rotation, so no need to check for
* that
* @param node the node that is in need of a right rotation
* @return the new root of the subtree after the rotation
*/
public AVLNode rightRotation(AVLNode node) {
    Node B = node.left;
```

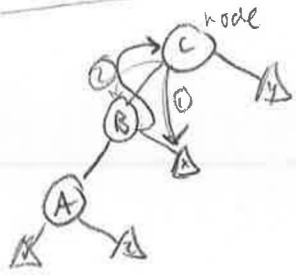
Rotate such that the nodes end up in their final position when it is complete. ***[Must be 2 lines]***

```
node.left = B.right;
B.right = node;
```

Update the nodes with correct heights and balance factors. ***[Must be 2 lines]***

```
updateNode(node);
updateNode(B);
```

```
    return B;
}
}
```



7) Quick Sort - Fill in the Blank Coding [15 points]

The following code is a recursive helper method implementing the quicksort algorithm taught in class. However, some lines are left open for you to implement. There will be boxes for you to fill in the lines of code missing. The boxes contain italicized instructions that you ***must*** follow. Do **not** write any helper methods; your work **must** be self contained within the boxes and should work with the lines of code that have already been provided.

```
/**
 * A recursive helper method for the quicksort algorithm.
 *
 * The algorithm should be implemented as in class except that the pivot selected will
 * always be the [redacted] element in the subarray (not randomized).
 *                FIRST
 * @param arr The array to sort using quicksort.
 * @param left The left bound of the current subarray. This bound is INCLUSIVE!
 * @param right The right bound of the current subarray. This bound is INCLUSIVE!
 */
private static void quickSort(int[] arr, int left, int right) {
```

*Write the base case for the method. **[3 or more lines]***

```
if (left >= right) {
    return;
}
```

```
    int pivot = arr[left];
    int i = left + 1;
    int j = right;
    while (i <= j) {
```

*Write the logic for moving the i index in this box. **[3 or more lines]***

```
while (i <= j && arr[i] <= arr[pivot]) {
    i++;
}
```

*Write the logic for moving the j index in this box. **[3 or more lines]***

```
while (i <= j && arr[j] >= arr[pivot]) {
    j++;
}
```

```
        if (i <= j) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }
    }
```

```
// Continued from previous page
```

Swap the pivot into the correct position in this box. ***[2 or more lines]***

```
int piv = arr[pivot];
arr[pivot] = arr[j];
arr[j] = piv;
```

Write the recursive call(s) necessary in this box.

```
quickSort(arr, left, j-1);
quickSort(arr, j+1, arr.length-1);
```

```
}
```

8) KMP - Failure Table Diagram [10 points]

Given the pattern **anabanana**, write the full failure table used by the KMP algorithm. You are not required to show the positions of i and j throughout each iteration: you just need to show the final failure table.

i	0	1	2	3	4	5	6	7	8
pattern[i]	a	n	a	b	a	n	a	n	a
failure[i]	0	0	1	0	1	2	3	2	3

This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.