

CSE 373 Practice Midterm Exam #3
(Section Handout #7)
ANSWER KEY

1. Big-Oh Analysis

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N)$
- d) $O(N^2)$

2. Java / Guava Collection Programming

```
public List<String> rankFamilies(Table<String, String, Integer> scores) {
    BiMap<String, Integer> totals = HashBiMap.create();
    for (String lastName : scores.rowKeySet()) {
        totals.put(lastName, 0);
        for (int points : scores.row(lastName).values()) {
            totals.put(lastName, totals.get(lastName) + points);
        }
    }

    Queue<Integer> pq = new PriorityQueue<Integer>(totals.values());
    List<String> winners = new LinkedList<String>();
    while (!pq.isEmpty()) {
        int points = pq.remove();
        String family = totals.inverse().get(points);
        winners.add(0, family);
    }
    return winners;
}
```

3. Java Class Programming for Collections

```
public class Car implements Comparable<Car> {
    private String make;           // such as "Toyota"
    private String model;          // such as "Camry"
    private int year;              // such as 2011
    private Color color;

    ...

    public int compareTo(Car other) {
        if (!make.equals(other.make)) {
            return make.compareTo(other.make);
        } else if (year != other.year) {
            return year - other.year;
        } else if (!model.equals(other.model)) {
            return model.compareTo(other.model);
        } else {
            return color.compareTo(other.color);
        }
    }

    public boolean equals(Object o) {
        if (o instanceof Car) {
            Car other = (Car) o;
            return make.equals(other.make) && model.equals(other.model) &&
                year == other.year && color.equals(other.color);
        } else {
            return false;
        }
    }

    public int hashCode() {
        return 13 * make.hashCode() +
            37 * model.hashCode() +
            57 * color.hashCode() +
            1337 * year;
    }
}
```

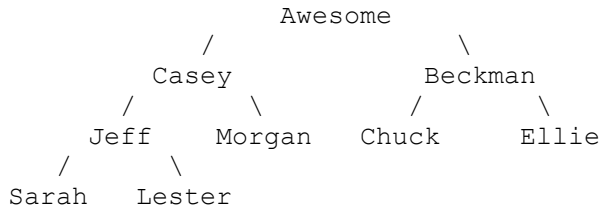
4. Hashing

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19													
	/		/		/		/		+		/		/		/		+		/		+		/		/		+		/		/	
				↓					↓					↓	↓					↓												
				444					47					-232	-34					97												
				↓																↓												
				-4																77												
				↓																												
				44																												

```
size      = 8
capacity  = 20
load factor = 0.4
```

5. Heaps

a) after all adds, final min-heap tree:

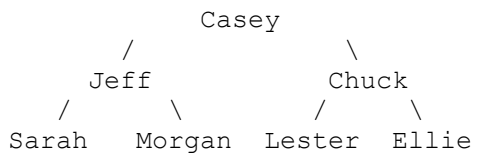


array:

```

0 1 2 3 4 5 6 7 8 9 10 11
[/, Awesome, Casey, Beckman, Jeff, Morgan, Chuck, Ellie, Sarah, Lester, /, ...]
  
```

b) after 2 removes, final min-heap tree:



array:

```

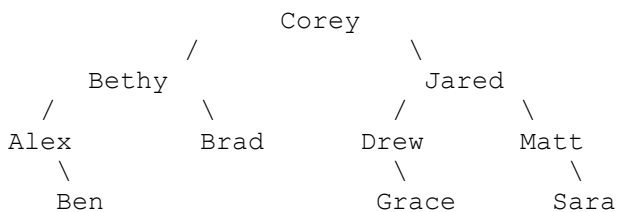
0 1 2 3 4 5 6 7 8 9
[/, Casey, Jeff, Chuck, Sarah, Morgan, Lester, Ellie, /, ...]
  
```

6. AVL Trees

a)

- adding Matt causes case-4 L rotation on Drew
- adding Bethy causes case-2 LR rotation on Drew
- adding Brad causes case-2 LR rotation on Bethy
- adding Ben causes case-1 R rotation on Jared

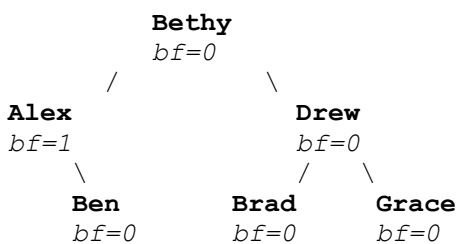
- after all adds, AVL tree:



b) and c)

- removing Sara causes case-2 LR rotation on Matt
- removing Corey (replaced by Drew) causes case-1 R rotation on Drew

after all removes, AVL tree:



7. Heap Priority Queue Implementation

```
public void removeInRange(E min, E max) {
    for (int i = 0; i < elements.length; i++) {
        if (elements[i] != null) {
            // remove from front, if any
            while (elements[i] != null && isBetween(elements[i].data, min, max)) {
                elements[i] = elements[i].next;
                size--;
            }

            // remove from rest of chain
            Node current = elements[i];
            while (current != null && current.next != null) {
                if (isBetween(current.next.data, min, max)) {
                    current.next = current.next.next;
                    size--;
                } else {
                    current = current.next;
                }
            }
        }
    }
}

// returns true if min <= value <= max
private boolean isBetween(E value, E min, E max) {
    return ((Comparable<E>) value).compareTo(min) >= 0 &&
        ((Comparable<E>) value).compareTo(max) <= 0;
}
```