CSE 373, Winter 2013
Midterm Exam, Wednesday, February 20, 2012

Name: _____

Quiz Section: _____ **TA:** _____

Student ID #: _____

Rules:

- You have **60 minutes** to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This test is open-book/notes. You may use any paper resources or textbooks you like.
- You may *not* use any computing devices, including calculators, cell phones, or music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- Please do not abbreviate code, such as writing ditto marks ("") or dot-dot-dot marks (...).
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your **Student ID** to a TA or instructor for your submitted exam to be accepted.

Good luck! You can do it!

Problem	Description	Earned	Max
1	Big-Oh		15
2	Collection Programming		20
3	Hashing		15
4	Heaps		15
5	AVL Trees		15
6	Collection Implementation		20
TOTAL	**Total Points**		**100**

1. Big-Oh

Give a tight bound of the runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable *N*. Write your answer on the right side.

Question | **Answer**

a)

```
int sum = 0;
for (int i = 1; i <= N + N; i++) {
    sum++;
}
for (int j = 1; j <= N * 2; j++) {
    sum += 5;
}
System.out.println(sum);
```

O(_____)

b)

```
int sum = 0;
for (int i = 1; i <= N - 5; i++) {
    for (int j = 1; j <= N - 5; j = j * 2) {
        sum++;
    }
}
System.out.println(sum);
```

O(_____)

c)

```
int sum = N;
for (int i = 0; i < 1000000; i++) {
    for (int j = 1; j <= i; j++) {
        sum += N;
    }
    for (int j = 1; j <= i; j++) {
        sum += N;
    }
    for (int j = 1; j <= i; j++) {
        sum += N;
    }
}
System.out.println(sum);
```

O(_____)

d)

```
List<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        list.add(0, i + j);
    }
}

int count = 0;
for (int i = 1; i <= 2 * N; i++) {
    if (list.contains(i)) {
        count++;
    }
}

System.out.println("done!");
```

O(_____)

e)

```
Set<Integer> set1 = new HashSet<Integer>();
for (int i = 1; i <= N; i++) {
    set1.add(i);
}

Set<Integer> set2 = new TreeSet<Integer>();
for (int i = 1; i <= N; i++) {
    set1.remove(i);
    set2.add(i + N);
}
System.out.println("done!");
```

O(_____)

2. Java / Guava Collection Programming

Write a method named **`highlyPaidWomen`** that returns a set of names of women who earn more money than their husbands. The method accepts two parameters: 1) a **marriage `BiMap`** from strings to strings, where each key/value pair represents the name of a husband and wife; and 2) a **salary `Map`** from strings to real numbers, where each key/value pair represents a person's name and that person's salary at his/her job. Your method should examine these collections and return a `Set` of names of all women who are employed and earn a salary that is greater than than their husband's salary. The set should be in sorted alphabetical order. Note that not every person has a job; if not, they will be absent from the salary map. If the wife is employed and the husband is not, by default she makes more money than her husband and should be included in the set.

For example, suppose your method is passed the following marriage bi-map and salary map:

- `{Fred=Wilma, Homer=Marge, Peter=Lois, Tarzan=Jane, Simba=Nala, Ricky=Lucy, ...}`
- ```
  {Lois=29.0, Fred=44.0, Nala=97.0, Marge=8.0,
   Peter=13.0, Lucy=11.0, Homer=5.0, Ricky=22.0}
  ```

your method would return the set `[Lois, Marge, Nala]` because Lois makes 29.0 to Peter's 13.0, Marge makes 8.0 to Homer's 5.0, and Nala makes 97.0 while Simba is unemployed. Wilma and Jane are not in the set because they are unemployed, and Lucy is not in the set because she makes less salary than Ricky.

Your code should run in O(N log N) time where N is the number of key/value pairs in the salary map. For the purposes of this problem, we will assume that the marriage bi-map is likely to be much larger than the salary map, therefore it is inefficient and inappropriate to loop over every single key/value pair in the bi-map. You may assume that every name in the salary map appears in the bi-map as a key or value.

You may assume that none of the collections passed, nor any of the keys/values in them, are `null`.
Do not construct any other auxiliary collections other than the set of names to return.
You should not modify any of the collections passed in.

  ```

2. Java / Guava Collection Programming (additional writing space)

3. Hashing

Simulate the behavior of a **hash map** as described in lecture. Assume the following:

- the hash table array has an initial capacity of **10**
- the hash table uses **linear probing** for collision resolution
- the hash function returns the integer key's value, mod the size of the table, plus any probing needed
- **rehashing** occurs at the *end* of an add where load factor is ≥ **0.6** and doubles the capacity of the hash table
- the table uses **lazy removal** and stores a special value "XX" for an array entry that has been removed

Draw an array diagram to show the final state of the hash table after the following operations. Leave a box empty if an array element is `null` or unused. Also write the **size, capacity, and load factor** of the final hash table.

Please *show your work*. You do not have to redraw an entirely new hash table after each element is added or removed, but since the final answer depends on every add/remove being correct, you may wish to redraw the table at various important stages (such as after any rehashing, and in its final state) to earn partial credit in case of an error.

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
map.put(12, 34);
map.put(99, 11);
map.put(72, 56);
map.put(42, 7);
map.put(999, 9);
map.put(72, 88);

map.remove(0);
map.remove(99);
if (map.containsKey(72)) {
    map.remove(12);
}

map.put(32, 42);
map.put(63, 9);
map.put(81, 18);
map.put(22, 33);
map.remove(42);
```

---

4. Heaps

Given the following integer elements:

- 9, 2, 5, 6, 0, 7, 1, 4, 3, 8

a) Draw the tree representation of the **binary min-heap** that results when all of the above elements are **added** (in the given order) to an initially empty heap. Circle the final tree that results from performing the additions. Also show the final **array representation** of the heap.

Please *show your work*. You do not have to draw an entirely new tree after each element is added or removed, but since the final answer depends on every add/remove being done correctly, you may wish to show the tree at various important stages, along with an indication of any bubble operations, to help earn partial credit in case of an error. Please **circle your answer** to make it clear what part of the page is supposed to be graded.

---

4. Heaps

b) Given your heap from part a), perform **2 remove-min operations** on the heap and draw its state afterward. Circle the tree that results after the two elements are removed.

5. AVL Trees

Given the following integer elements:

- 8, 1, 0, 6, 9, 7, 4, 2, 5, 3

a) Draw the AVL tree that results when the above elements are **added** (in the given order) to an empty AVL tree.

Please *show your work*. You do not have to draw an entirely new tree after each element is added or removed, but since the final answer depends on every add/remove being done correctly, you may wish to show the tree at various important stages, along with an indication of any rotations performed, to help earn partial credit in case of an error. Please **circle your answer** to make it clear what part of the page is supposed to be graded.

---

5. AVL Trees

b) Draw the AVL tree from part (a) after all of the following elements are **removed**. Recall that our BST remove algorithm chooses the *leftmost element from the right subtree* if necessary on removal of a node.

- 3, 6, 8, 1

6. Hash Map Implementation

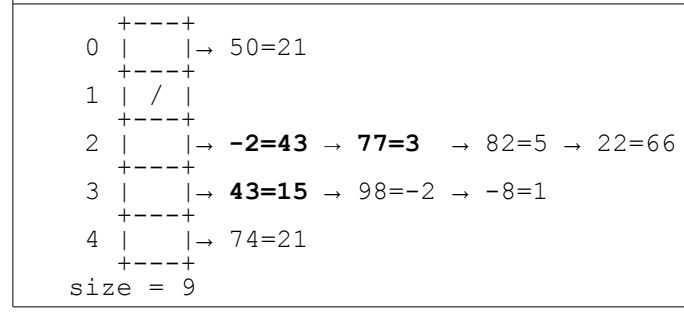
On your homework, you implemented a class called `HashMap`, a map of key/value pairs implemented using a hash table with separate chaining. Assume that the class is implemented in the following way:

```
public class HashMap<K, V> implements Map<K, V> {
    private Node[] elements;
    private int size;
    ...

    private int hash(K key) {...}

    private class Node {
        private K key;
        private V value;
        private Node next;
        ...
    }
}
```

Add a method to this class named `trimChains` that accepts an integer *k* as a parameter and ensures that the linked list of nodes in every index of the internal hash table is no longer than *k* nodes at most. If a given linked list is longer than k elements, you should shorten it by removing nodes from the front of the chain. For example, if a map named `map` has inner hash table storing the linked lists in the picture at left, and the client makes the call of `map.trimChains(2);`, your method should change the inner hash table to store the lists in the picture at right.

`map` before	**after `map.trimChains(2);`**
	

If the value of *k* passed is 0 or negative, you should completely empty each chain of all nodes. Your map's `size` field should store the correct value after your method, so if you remove any nodes, update the size. Do not recreate the internal hash table or copy it entirely. You don't need to consider load factor or rehashing on this problem.

You should not create any arrays or temporary data structures. This method should run in O(*N*) time, where *N* is the total number of nodes in the map. For full credit, do not walk the entire length of any linked list chain more than once. Don't call any of the existing methods on your hash map, since they would hurt the efficiency of your method.

Write your answer on the next page.

6. Hash Map Implementation (writing space)

Extra writing space