CIS 338 (Fall 2012)

Exam 1, 10/8/12

Name (sign)

Name (print)

email

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	

Question 1

For each operation below, give its (a) worst-case and (b) average (expected, amortized, etc.) asymptotic running-time. Use abbreviations from the table.

A	$\sim c$	**E**	$\sim c\,\mathbf{N}$	**J**	$\sim c\,\mathbf{N}^2$
B	$\sim c\ lg\ \mathbf{n}$	**F**	$\sim c\,\mathbf{n}\ lg\ \mathbf{n}$	**K**	$\sim c\ lg\ \mathbf{m}$
C	$\sim c\ lg\ \mathbf{N}$	**G**	$\sim c\,\mathbf{N}\ lg\ \mathbf{N}$	**L**	$\sim c\,\mathbf{m}$
D	$\sim c\,\mathbf{n}$	**H**	$\sim c\,\mathbf{n}^2$	**M**	$\sim c\,\mathbf{m}\ lg\ \mathbf{m}$

rank on a sorted array using binary search

append to an **`ArrayList`**

dequeue from a **`LinkedQueue`**

iterator on a **`ArrayStack`**

find on a **`ListUnionFind`** (assume all symbol table operations are unit cost)

connect on a **`TreeUnionFind`** (same assumption)

get on a **`SequenceSymbolTable`**

keys on a **`LinkedListIntegerTable`**

put on an **`ExternalHashTable`**

size on an **`InternalHashTable`**

Question 2

What is the **rank** of **65** in the sorted array below?
Draw a line through each array element that would be tested in a binary search to determine that rank.

97
73
71
67
61
59
57
51
47
41
37
31
29
23
17
11

Question 3

Complete the implementation of rank below using binary search.

```
/**
 * How many values in a sorted array are less than a given value?
 *
 * Note if d is in a then d == a[rank].
 *
 * @param d the given value.
 * @param a the sorted (from smallest to largest) array of values.
 * @return the number of values in a that are less than d.
 */
public static int rank (double d, double[] a) {
    return rank (d, a, 0, a.length);
}

/**
 * How many values in a sorted array are less than a given value?
 * The rank is guaranteed to be in a given range: [lo, hi].
 * That is:    lo <= rank <= hi.
 *
 * Note if d is in a then d == a[rank].
 *
 * @param d the given value.
 * @param a the sorted (from smallest to largest) array of values.
 * @param lo the bottom of the range.
 * @param hi the top of the range.
 * @return the number of values in a that are less than d.
 */
private static int rank (double d, double[] a, int lo, int hi) {



}
```

Question 4

Draw an **`ArrayList`** and a **`LinkedList`** after the operations below have been performed. The list are initially empty. The initial capacity of the **`ArrayList`** is 5.

```
append("A");
prepend("B");
append("C");
prepend("D");
append("E");
remove();
```

Question 5

Complete the following list-based **stack** implementation.

```
public abstract class SomeStack<Item> implements Stack<Item> {
    private final List<Item> list = new LinkedList<Item>();

    @Override public boolean isEmpty() {




    }
    @Override public int size() {




    }
    @Override public final void push (Item item) {




    }
    @Override public final Item pop () {




    }
    @Override public final Item peek () {




    }
    @Override public final Iterator<Item> iterator() {




    }

}
```

Question 6

Show the contents of the *internal* **hash table** below after the following operations. The **`hash()`** and **`rehash()`** values of the **`key`**s are shown in the table. (The hash table has 11 entries. Remember that the first index is 0.)

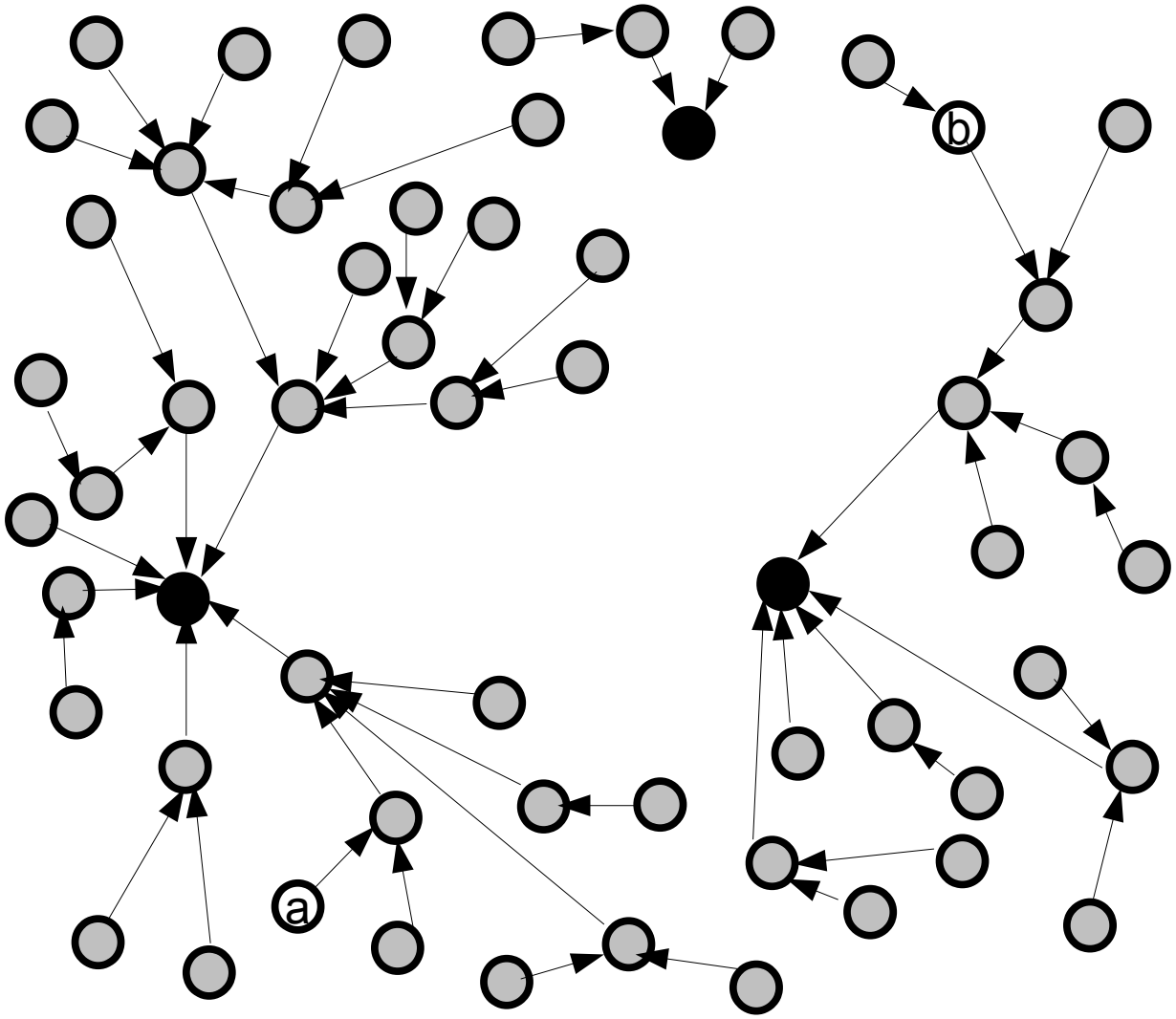
key	hash	rehash
a	3	5
b	2	2
c	4	4
d	3	1
e	2	3
f	4	1
g	1	3
h	2	3
i	1	1

```
put(d,0); put(e,1); put(a,2); put(d,3);
put(g,4); put(b,5); delete(e); put(h,6);
put(f,7); put(c,8); put(i,9); delete(b);
```

key	value

Question 7

Below is a representation of a WeightedTreePathCompressionUnionFind data-structure. Show how executing **`connect(a, b)`** would modify this data-structure.



Question 8

Complete the following tree-based UnionFind implementation.

```
public class TreeUnionFind<Vertex> extends AbstractUnionFind<Vertex>{
    protected final SymbolTable<Vertex, Vertex> parent;

    /**
    * Create a new Union Find data structure.
    */
    public TreeUnionFind () {
        parent = new SequenceSymbolTable<Vertex, Vertex>();
    }
    @Override public void union (Vertex v, Vertex w) {



    }
    @Override public Vertex find (Vertex v) {



    }
}
```

Question 9

Complete the three methods below to implement a Queue of Item's.

```
public class Queue {
  private Item[] data = new Item[1];
  private int count = 0; // number of items in the queue
  private int first = 0; // index of the oldest item
  private int index (int i) { // index of the i-th item
      return (i+first+data.length) % date.length; }
  public void enqueue (Item item) {









  }
  public Item dequeue () {









  }
  private void resize (int max) {
    assert count < max;









  }
}
```