Examination 1

CSC301 Algorithms and Data Structures

24 March 2015

1. Complete the following table. Approximate values will suffice. (Approximate by truncating.)

N	$\log_2 N$	N^2	$N \log_2 N$
1	0	1	0
10	≈ 3	10^2	$\approx 3 \cdot 10$
10^2	≈ 6	10^4	$\approx 6 \cdot 10^2$
10^3			
10^6			
10^9			
10^{12}			

N	$\log_2 N$	N^2	$N \log_2 N$
1	0	1	0
10	≈ 3	10^2	$\approx 3 \cdot 10$
10^2	≈ 6	10^4	$\approx 6 \cdot 10^2$
10^3	≈ 10	10^6	$\approx 10 \cdot 10^3$
10^6	≈ 20	10^{12}	$\approx 20 \cdot 10^6$
10^9	≈ 30	10^{18}	$\approx 30 \cdot 10^9$
10^{12}	≈ 40	10^{24}	$\approx 40 \cdot 10^{12}$

2. There are places in our textbook where the authors describe opportunities to reduce the time required for a computation by 30%. However, in other places they emphasize that our ability to develop, analyze, and select algorithms can have a much bigger impact. Explain.

Sometimes choosing an algorithm poorly will make it impossible to solve the problem. Finding the right algorithm might be the prerequisite to putting a solution in reach.

3. Within the selection sort algorithm there is an algorithm that searches. There is also a search inside of the insertion sort. How do the two searches

differ?

The search in the selection sort proceeds from left to right through the still unordered part of the list. It must examine every element in the unordered part to the list in order to identify the element with the smallest value.

The search in the insertion sort proceeds from right to left through the already sorted part of the list. It does not need to examine every element in this part of the list. It quits as soon as it finds the right place in which to insert the next element.

4. What are the three defining properties of an equivalence relation?

An equivalence relation is reflexive, symmetric, and transitive.

For all elements a, b, and c:

- a is related to a
- If a is related to b, then b is related to a.
- If a is related to b and b is related to c, then a is related to c.

5. What is the significance of <ElementType> in this code?

A programmer will substitute the name of some specific class (a specific type) for <ElementType> when creating an instance of the Bag class. In this way, the programmer specifies what kinds of objects the Bag may hold.

```
package bag;

import java.util.Iterator;

public class Bag<ElementType> implements Iterable<ElementType> {

    private Node<ElementType> anchor;
    private int size;

    public Bag() {
        this.anchor = null;
        this.size = 0;
    } // Bag()
    public boolean isEmpty() {
        return this.size == 0;
    } // isEmpty()

    public int size() {
        return this.size();
    } // size()

    public void add( ElementType value ) {
        this.anchor = new Node( value, this.anchor );
        this.size++;
    } // add( ElementType )

    public Node getAnchor() {
        return this.anchor;
    } // getAnchor()

    @Override
    public Iterator<ElementType> iterator() {
        return new BagIterator(this);
    } // iterator()

} // Bag()
```

6. I did not have to type every line of this code. What part was NetBeans able to fill in for me?

NetBeans wrote the **package** statement and the getters (the accessors—getValue() and getNext()). It also anticipated my needs for closing braces and parentheses.

```
package bag;

public class Node<ElementType> {
    private final ElementType value;
    private final Node next;

    public Node( ElementType value ) {
        this.value = value;
        this.next = null;
    } // Node( ElementType )

    public Node( ElementType value, Node next ) {
            this.value = value;
            this.next = next;
        } // Node( ElementType, Node )

        public ElementType getValue() {
            return this.value;
        } // getValue()

        public Node getNext() {
            return this.next;
        } // getNext()
    } // Node
```

7. Complete the definition of the BagIterator class.

```
package bag;

import java.util.Iterator;

public class BagIterator<ElementType> implements Iterator<ElementType>
    private final Bag bag;
    private Node<ElementType> currentNode;

    public BagIterator( Bag bag ) {
        this.bag = bag;
        this.currentNode = this.bag.getAnchor();
    } // BagIterator( Bag )

    @Override
    public boolean hasNext() {
     // This is a stub method---complete its definition.
      return false;
    } // hasNext()

    @Override
    public ElementType next() {
      // This is a stub method---complete its definition.
      return null;
    } // next()
} // BagIterator
```

```
    @Override
    public boolean hasNext() {
        return this.currentNode != null;
    } // hasNext()

    @Override
    public ElementType next() {
        ElementType value = this.currentNode.getValue();
        this.currentNode = this.currentNode.getNext();
        return value;
    } // next()
```

8. Here is code that creates a Bag and adds several integers to it. Add code that will print the values of the integers in the Bag.

   (The Bag class **implements** the Iterator interface.)

```
Bag<Integer> moonLandings = new Bag<>():
moonLandings.add( 11 );
moonLandings.add( 12 );
moonLandings.add( 14 );
moonLandings.add( 15 );
moonLandings.add( 16 );
moonLandings.add( 17 );
```

```
for( int n : moonLandings ) {
  System.out.println( n );
} // for

// or, we could do it this way...
Iterator<Integer> iterator = moonLandings.iterator();
while( iterator.hasNext() ) {
  int n = iterator.next();
  System.out.println( n );
} // while
```

9. Complete the code for the selection sort.

```
    public static int positionOfMinimum(int[] data, int i) {
        int bestGuessSoFar = i;
        for (int j = i; j < data.length; j++) {
            if (data[j] < data[bestGuessSoFar]) {
                bestGuessSoFar = j;
            } // if
        } // for
        return bestGuessSoFar;
    } // positionOfMinimum( int [], int )

    public static void swap(int[] data, int i, int j) {
        int temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    } // swap( int [], int, int )

    public static void selectionSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
          // Code is needed here.
        } // for
    } // selectionSort( int [] )
} // InOrder
```

```
    public static void selectionSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            int j = positionOfMinimum(data, i);
            swap(data, i, j);
        } // for
    } // selectionSort( int [] )
```

10. Complete the code for the insertion sort.

```
    public static int insertionPoint(int[] data, int i) {
        int j = i;
        while (j > 0 && data[j - 1] > data[i]) {
            j--;
        } // while
        return j;
    } // insertionPoint( int [], int )

    public static void insert(int[] data, int i, int j) {
        int temp = data[i];
        for (int k = i; k > j; k--) {
            data[k] = data[k - 1];
        } // for
        data[j] = temp;
    } // insert( int [], int, int )

    public static void insertionSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
```

```
        // Code is needed here.
    } // for
} // insertionSort( int [] )
```

```
public static void insertionSort(int[] data) {
    for (int i = 0; i < data.length; i++) {
        int j = insertionPoint(data, i);
        insert(data, i, j);
    } // for
} // insertionSort( int [] )
```