

Family Name: Other Names:

Student ID: Signature.....

COMP 103 : Test 5

2021, Feb 5

Instructions

- Time allowed: **50 minutes**
- Attempt **all** questions. There are 30 marks in total.
- Write your answers in this test paper and hand in all sheets.
- If you think some question is unclear, ask for clarification.
- Brief Java documentation is provided with the test
- This test contributes 10% of your final grade
- You may use dictionaries.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions

Marks

1. Graphs

[10]

2. Binary Search

[10]

2. Heaps

[10]

TOTAL:

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. Traversing Graphs**[10 marks]**

You are writing a program to investigate interactions on a social media platform, such as Twitter. You are using a graph to keep track of the interactions between accounts.

Your program stores information about the accounts involved in a Set of `TwitterAccount` objects.

```
private Set<TwitterAccount> allTwitterAccounts; // all accounts being investigated
```

- Each `TwitterAccount` object contains a Set of other `TwitterAccounts` that it has interacted with.
- `TwitterAccount` is iterable, so you can iterate through all of an account's interactions with a for-each loop.
 - E.g. `for(TwitterAccount interactions : account)`
- You do not need to know about any of the other fields or methods in `TwitterAccount`.

(a) **[5 marks]** Complete the following `countConnected` method which is given a `TwitterAccount` and should count the total number of accounts that can be reached by following the graph of interactions.

The method should use the visited set to keep track of which accounts it has already processed.

```
public int countConnected(TwitterAccount account){
    Set<TwitterAccount> visited = new HashSet<TwitterAccount>();
    return countConnected(account, visited );
}

public int countConnected(TwitterAccount account, Set<TwitterAccount> visited){

}

}
```

(Question 1 continued on next page)

(Question 1 continued)

(b) [2 marks] Describe, in your own words, what it means for a graph to be disconnected.

(c) [3 marks] Complete the following `isConnected(...)` that checks to see if the graph of interactions is connected.

You may need to use the method you wrote above.

```
public boolean isConnected(TwitterAccount start){
```

```
}
```

Question 2. Binary Search**[10 marks]**

This question concerns the binary search algorithm presented below.

```

public int indexOf ( String value , List < String > data ) {
    int low = 0;
    int high = data . size ( ) ;
    while ( low < high ) {
        int mid = ( low + high ) / 2;
        int comp = value . compareTo ( data . get ( mid ) );

        // Found the item -- return the index
        if ( comp ==0) return mid ;

        if ( comp < 0) {
            // Check lower half
            high = mid ;
        } else {
            // Check upper half
            low = mid + 1;
        }
    }
    return -1;
}

```

(a) **[2 marks]** True or False: Binary Search works on both sorted and unsorted lists.

(b) **[3 marks]** Explain why the binary search algorithm has a cost of $O(\log n)$.

(Question 2 continued)

(c) [5 marks] Rewrite the method on the previous page so that instead of returning -1 if it can't find an item, it instead returns the index that item *would* exist at if it were in the list.

For example, if given the list below, `findIndex("E", list)` should return 4 (where "E" should be inserted), rather than -1.

"A"	"B"	"C"	"D"	"F"	"G"	"H"	"I"	"J"
0	1	2	3	4	5	6	7	8

```
public int findIndex(String value, List<String> data) {
```

```
}
```

Question 3. Heaps**[10 marks]**(a) **[2 marks]** What is the defining property of a partially ordered tree?
(b) **[2 marks]** A heap is a partially ordered tree stored in an array. If a node is stored at index 3, what are the indices of its left and right children (if they exist)?
(c) **[6 marks]** Suppose the values below were added, in the order shown, to a heap that is initially empty.**A-34, B-70, C-12, D-54, E-36, F-99, G-04**

Draw the partially ordered tree that represents the heap after each value is added.

- The value of the item is represented by the number, so A has a value of 34, for example.
- The heap should be a **max** heap, so F-99 should end up as the root of the tree.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Documentation

Brief, simplified specifications of some relevant Java collection types and classes.

Note: E stands for the type of the item in the collection.

interface *Collection* <E>

```
public boolean isEmpty()           // cost: O(1) for all standard collection classes
public int size()                 // cost: O(1) for all standard collection classes
public void clear()
public boolean add(E item)
public boolean contains(Object item)
public boolean remove(Object element)
```

interface *List* <E> **extends** *Collection* <E>

// Implementations: ArrayList

```
public boolean isEmpty()
public int size()
public void clear()
public E get(int index)           // cost: O(1)
public E set(int index, E element) // cost: O(1)
public boolean contains(Object item) // cost: O(n)
public void add(int index, E element) // cost: O(n) (unless index is close to the end.)
public E remove(int index)       // cost: O(n) (unless index is close to the end.)
public boolean remove(Object element) // cost: O(n)
```

interface *Set* **extends** *Collection* <E>

// Implementations: HashSet, TreeSet

```
public boolean isEmpty()
public int size()
public void clear()
public boolean add(E item)         // cost: O(1) for HashSet, O(log(n)) for TreeSet
public boolean contains(Object item) // cost: O(1) for HashSet, O(log(n)) for TreeSet
public boolean remove(Object element) // cost: O(1) for HashSet, O(log(n)) for TreeSet
```

interface *Map* <K, V>

// Implementations: HashMap, TreeMap

```
public V get(K key)               // cost: O(1) for HashMap, O(log(n)) for TreeMap
public V put(K key, V value)      // cost: O(1) for HashMap, O(log(n)) for TreeMap
public V remove(K key)           // cost: O(1) for HashMap, O(log(n)) for TreeMap
public boolean containsKey(K key) // cost: O(1) for HashMap, O(log(n)) for TreeMap
public Set <K> keySet()          // cost: O(1)
public Collection <V> values()  // cost: O(1)
// (get returns null if the key is not present)
// (get put and remove return the old value, if any)
```

```
interface Queue<E> extends Collection<E>
    // Implementations: ArrayDeque, LinkedList, PriorityQueue
    public boolean isEmpty()
    public int size()
    public void clear()
    public E peek () // cost: O(1) for ArrayDeque, LinkedList, O(1) for PriorityQueue
    public E poll () // cost: O(1) for ArrayDeque, LinkedList, O(log(n)) for PriorityQueue
    public boolean offer (E element) // cost: O(1) for ArrayDeque, LinkedList, O(log(n)) for PriorityQueue
    // (peek and poll return null if the queue is empty)
```

```
class Stack<E> implements Collection<E>
    public boolean isEmpty()
    public int size()
    public void clear()
    public E peek () // cost: O(1)
    public E pop () // cost: O(1)
    public E push (E element) // cost: O(1)
    // (peek and pop return null if the queue is empty)
```

```
class Collections
    public void sort (List<E> list); // cost = O(n log(n)), but O(n) if almost sorted
    public void sort (List<E> list, (E e1, E e2)->{...}); // cost = O(n log(n)), but O(n) if almost sorted
    public void swap(List<E> list, int i, int j); // cost = O(1)
    public void reverse (List<E> list); // cost = O(n)
    public void shuffle (List<E> list); // cost = O(n)
```

```
interface Comparable<E> // All Comparable objects have a compareTo method:
    public int compareTo(E other);
        // returns
        // -ve if this comes before other;
        // +ve if this comes after other,
        // 0 if this and other are the same
```
