Logbook

Bradley Pratt - Computer Games Programming U1664020314

Algorithms Processes and Data

Week 1-2 :

```
package intArrays;

import java.util.Arrays;

public class CleverRandomListing extends RandomListing {

	public CleverRandomListing (int size) {
		super(size);
	}

	/**
	 * The purpose of this method is to rebuild an array in a completely random
order
	 * Passes an array in from the SortedListing class.
	 */
	protected void randomise() {

		for (int index = 0; index < getArray().length; index++) {
			int randomArray = getRandomIndex(); // Uses the getRandomIndex
method to randomise the array index

			int newInt = getArray()[randomArray];
			getArray()[randomArray] = getArray()[index]; // Changes the
grabbed array to randomise its index
			getArray()[index] = newInt; // Builds the array using its new
index
		}
	}

    public static void main(String[] args) {
	RandomListing count = new CleverRandomListing(50); // create a new list, as
long as the specified length.
	System.out.println(Arrays.toString(count.getArray())); // prints the array
to the console
    }
}
```

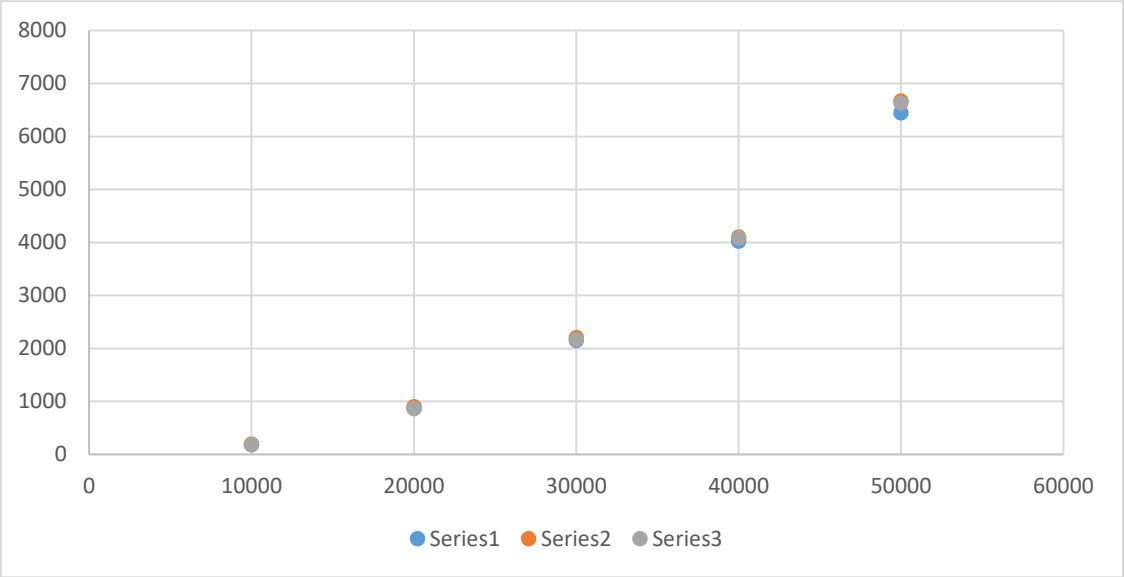
The tests for this class showed that is more efficient than the standard sorting class, with a testMillionSize taking 96423 milliseconds. In SimpleRandomTesting testMillionSize took 261148 milliseconds in my last test; proving the above shown method is more efficient.

Week 3-4

```
	/**
	 * Swaps the specified elements within the array
	 * @param array the array which is passed into the method
	 * @param index1 the index which needs to be swapped with index2
	 * @param index2
	 */
	public static <T> void swap(T[] array, int index1, int index2) {
		T objectOne = array[index1];
		T objectTwo = array[index2];

		array[index1] = objectTwo; //Uses the defined first position and
places "objectTwo" there
		array[index2] = objectOne; //Uses the defined second position and
places "objectOne" there
	}

	/**
	 * The purpose of max is to find the largest element in between index1 and
index2.
	 * @param array is the array that is passed in
	 * @param index1 is the first index, which elements before it may be
ignored
	 * @param index2 is the second index location, which elements after it may
be ignored
	 * @return returns the largest element
	 */
    public static <T> String max(String[] array, int index1, int index2) {
	  int index = 0;
	  int elementLength = array[0].length();
	  System.out.println();
	  for (int i = 0; i < array.length; i++) {
		if (i >= index1 && i <= index2) {
			if (array[i].length() > elementLength) {
				index = i;
				elementLength = array[i].length();
			}
		}
	  }
	  return array[index];
    }


	@Test
	public void testMax() {
		String[] nameTest = {"Hugh", "Andrew", "Ebrahim","Diane","Paula",
"Simon"};
		assertEquals("Ebrahim",GenericMethods.max(nameTest, 0, 5));
	}
```

The tests class for the swap method shows that the elements are successfully swapped, using the array 1,2,3,4,5 and adding index1 as 1, and index2 as 2 showed that the array became 1,3,2,4,5 as expected.

Week 5



(10,000) = 180

(20,000) = 890

(30,000) = 2150

(40,000) = 4000

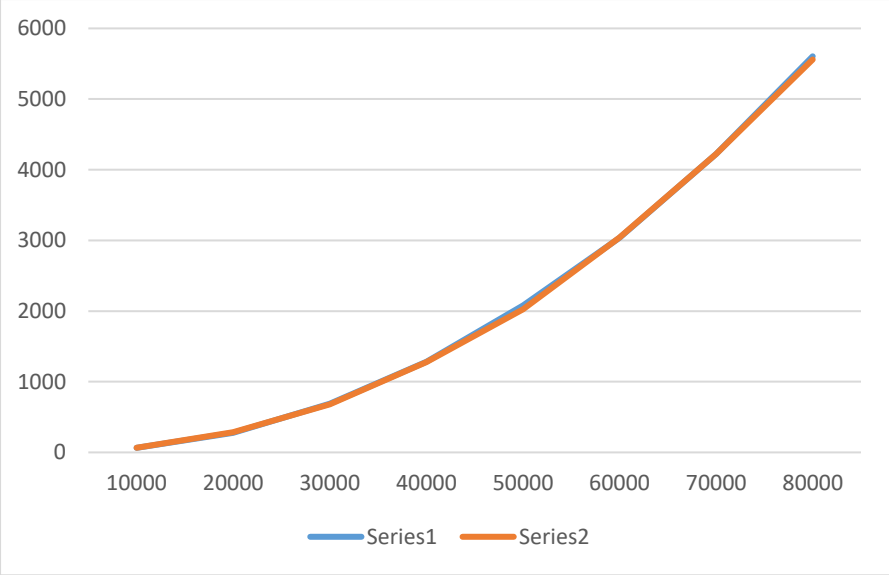
(50,000) = 6500

I ran the test three times and above are the results I got, the formulas show the average result found between the three tests. A function couldn't be found since there doesn't seem to be a running trend that would allow you to predict the next result with reasonably accuracy.

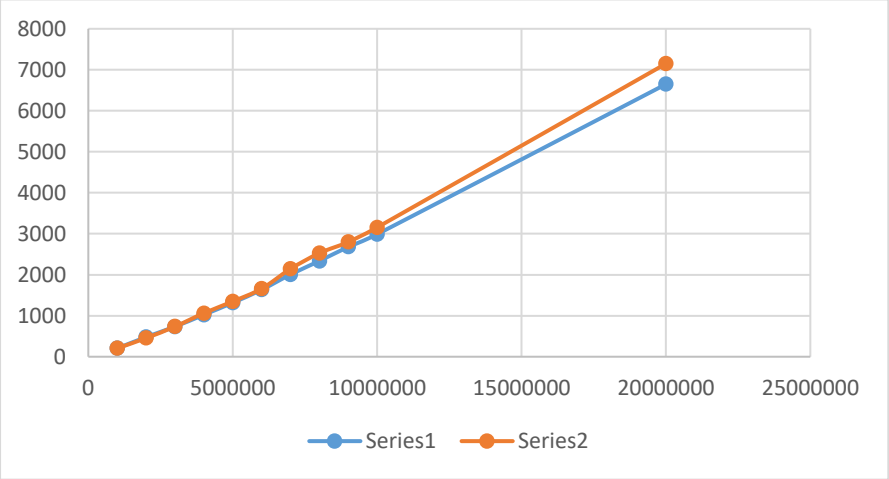
```
	/**
	 * Method for the SelectionSort
	 */
	public void sort(T[] array) {
	for (int i = 0; i < array.length; i++) {
		int minIndex = i;
		for (int j = i + 1; j < array.length; j++) {
			if (array[j].compareTo(array[minIndex]) < 0) {
				minIndex = j;
			}
		}
		if (minIndex != i) {
			T temp = array[i];
			array[i] = array[minIndex];
			array[minIndex] = temp;
		}
	}
}
```

The SelectionSort algorithm builds the array first and sorts it as it is built. The algorithm checks the value of each element of the array as it is inputted, if the value of the element at the position of j is greater than the element at the position of minIndex, then j is checked if it is not equal to i then i is replaced with j.

```
        /**
         * Method for the quicksort
         */
        private void sort(T[] array,int from,int to) {
              if (from < to) {
                    int pivotIndex = from;
                    int highIndex = to;
                    int lowIndex = pivotIndex;
                    T pivot = array[(highIndex + lowIndex) / 2];

              do { //Runs a do-while loop so that the method is ran whilst the
conditions are true
                  while (array[lowIndex].compareTo(pivot) < 0) lowIndex++;
//Increases the lowIndex amount by the amount of elements before the pivot
                  while (pivot.compareTo(array[highIndex]) < 0) highIndex--;
//Reduces the highIndex amount by the amount of elements above it, meaning
elements above the pivot are ignored

                  if (lowIndex <= highIndex) { //Checks the size of the element to
see if it can be swapped
                        T temp = array[lowIndex]; //Gets the lowIndex and places it in
the generic temp
                        array[lowIndex] = array[highIndex]; //Moves the smaller
element to the higher element
                        array[highIndex] = temp; //Changes the the value of highIndex
to the temp
                        lowIndex++;
                        highIndex--;
                  }

              } while (lowIndex <= highIndex); //Runs the do while this is true

              sort(array, from, highIndex);//Reruns the do-while loop with the new
pivot
              sort(array, lowIndex, to);
               }
    }
```



10000	64.472	65.462
20000	278.786	284.874
30000	690.19	678.444
40000	1281.19	1275.674
50000	2079.638	2026.574
60000	3033.605	3042.342
70000	4222.341	4223.89
80000	5605.172	5559.824



1000000	204.888	204.813
2000000	479.814	455.249
3000000	734.502	735.89
4000000	1023.676	1062.652
5000000	1316.623	1347.981
6000000	1633.81	1659.239
7000000	2005.972	2142.408
8000000	2335.258	2526.845
9000000	2679.885	2798.874
10000000	2988.025	3154.712
20000000	6648.698	7149.259

I ran two graphs, the first being the results selection sort and the second graph being the results for the quick sort. The selection sort doesn't seem to maintain a trend per each 10000. The quick sort shows an increase of 300 per 1000000 and overall looks a great deal more efficient than the selection sort algorithm.

Week 6

```
        private Node<T> head = null;
        private Node<T> tail = null;
        private int noOfNodes = 0;


            @Override
        public void add(int index, T value) throws ListAccessError {
            Node<T> addNode = new Node<T>(value); //Defines the value as a new
"addNode"
            if (isEmpty()) { //If the list is empty it adds it at the head
              head = addNode;
              tail = head;
            } else {
               Node<T> node = head;
               for (int i = 0; i < index-1; i++) { //Traverse through each node but
-1 to set next later
                     node = node.getNext();
               }
               node.setNext(addNode); //Continuation of the -1 so the next position
can be set
               node = addNode; //Sets the node to addNode from the beginning
            }
            noOfNodes++;
        }

        @Override
        public T remove(int index) throws ListAccessError {
        Node<T> node = head;
        Node<T> temp = getNode(index);
            if (isEmpty()) { //If the list is empty it adds it at the head
               throw new ListAccessError("Index out of bounds");
            } else {
               if (index < 0 || index >= noOfNodes) { // invalid index
                     throw new ListAccessError("Index out of bounds");
               }
               for (int i = 0; i < index-1; i++) { //Traverse through each node but
-1 to set next later
                     node = node.getNext(); //Continuation of the -1 so the next
position can be set
               }
               node.setNext(temp.getNext());
            }
            noOfNodes--;
            return node.getValue();
        }

        @Override
        public T get(int index) throws ListAccessError {
               return getNode(index).getValue();
        }
    private Node<T> getNode(int index) throws ListAccessError {
        if (index < 0 || index >= noOfNodes) { // invalid index
               throw new ListAccessError("Index out of bounds");
        }
        Node<T> node = head; // start at head of list
        for (int i = 0; i < index; i++) { // walk through list to desired index
```

```
            node = node.getNext(); // by following next references
        }
        return node; // return the node at the required index
    }
```

The test I used for the get and add is the test class used in the model test class, I did however modify it for the remove method as shown here:-

```
    @Test
    public void testRemoveTail() throws ListAccessError {
            SinglyLinkedList<Integer> list = new SinglyLinkedList<Integer>();
            list.add(0, 5);
            list.add(1, 7);
            list.add(2, 23);
            list.add(3, -6);
            list.add(4, 0);
            list.add(5,42);
            list.remove(5);
            thrown.expect(ListAccessError.class);
            thrown.expectMessage("Index out of bounds");
            list.get(5);
    }

    @Test
    public void testRemove() throws ListAccessError {
            SinglyLinkedList<Integer> list = new SinglyLinkedList<Integer>();
            list.add(0, 5);
            list.add(1, 7);
            list.add(2, 23);
            list.add(3, -6);
            list.add(4, 0);
            list.add(5,42);
            list.remove(3);
            assertEquals(new Integer(42),list.get(4));
    }
```

Using the model test class and the modified test class for the remove method it showed that the program ran with no errors.

Both the remove and add methods work by going through each node until the index is reached, taking away by one so the getNext() can be used and and then setting the next variable outside of the for loop.

Week 7

```
package binaryTree;

public class BinaryTree<T extends Comparable<? super T>> implements BTree<T> {
	TreeNode<T> root;

	@Override
	public void insert(T value) {
		if (root == null) {
				root = new TreeNode<T>(value);
			} else if (value.compareTo(value()) < 0) {
				root.left().insert(value);
			} else {
				root.right().insert(value);
			}
	}

	@Override
	public T value() {
		return root.value;
	}

	@Override
	public BTree<T> left() {
		return root.left;
	}

	@Override
	public BTree<T> right() {
		return root.right;
	}

}
```

```
class BinaryTreeTest {

	@BeforeAll
	static void setUpBeforeClass() throws Exception {

	}

	@AfterAll
	static void tearDownAfterClass() throws Exception {

	}

	@BeforeEach
	void setUp() throws Exception {

	}

	@AfterEach
	void tearDown() throws Exception {

	}

	@Test
	void test() {
		BinaryTree<Integer> tree = new BinaryTree<>();
		tree.insert(1);
		tree.insert(0);
		tree.insert(2);

		Integer topValue = 1;
		Integer leftValue = 0;
		Integer rightValue = 2;
		assertEquals(topValue, tree.value());
		assertEquals(leftValue, tree.left().value());
		assertEquals(rightValue, tree.right().value());
	}
}
```

Week	Overall	Documentation	Structure	Names	Tests	Function
1&2 Search Timer	C	C	B	B	D	B
3&4 Generic Swap	B	C	B	B	B	A
5 Sorting	A	A	B	A	A	A
6 Linked Lists	A	B	A	A	A	A
7 Binary Trees	C	F	B	C	B	A

Week 1&2:

I believe for this week I deserve a grace C because my testing was weak and only used a System.out to test the output was correct, but in conjunction to this I believe I deserve a C because the code does do what is required. I think the documentation is good, it is concise due to the fact I was unsure of how to go into greater detail on the subject and it probably could have used more but I think it shows that the code is functional.

Week 3&4:

This week I believe I deserve a grade B, my documentation is improved from the prior week in regards to the commenting in the code. I believe the structure and naming of my code is viable. I believe the testing is good since I have made a test class which is shown and it returns positively. I also believe the functionality of this week is good since it does what is required efficiently.

Week 5:

For this week I believe I deserve an A mainly due to the fact I made the sorting algorithms and because of the documentation alongside it. I ran each test a number of times and included the results and placed them into the graphs and explained the results.

Week 6:

For this week I believe I deserve a grade A because I believe the testing and functionality of my code is good. The tests clearly show the code functions as it should and my documentation in the logbook show how the code works and my comments show how it works as the code runs. I also believe the structure and naming is good since it is clear to understand what does what by reading the names alone.

Week 7:

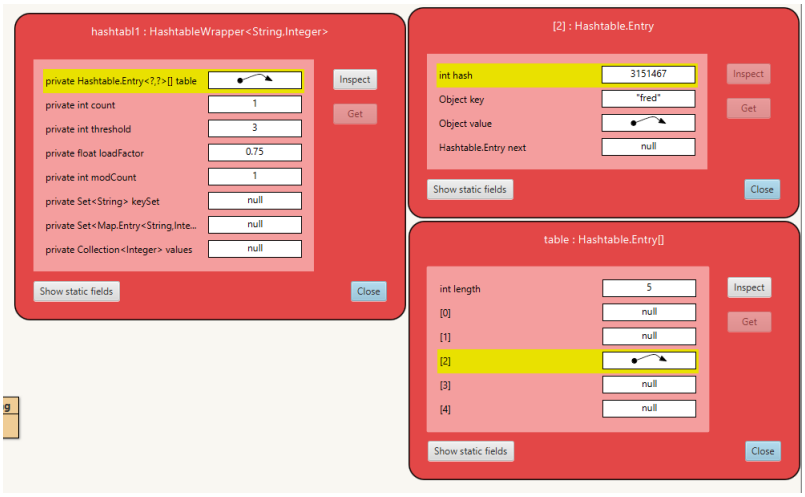
For this week I believe I deserve a grade C, this week fell really weak on documentation, but I believe it was strong in every other aspect. I made a test class which shows that the code functions correctly which is why I believe it is strong in these regards and that overall the week deserves a C.

Week 8

The hashtableWrapper in week 8 uses modular arithmetic to sort the positions of objects within the hashtable. The way it works out where to place these objects is to use the following equation of “object hash” % “length of hashtable”. This is an efficient way to store the objects of the hash table since conflicts are unlikely since the hash is unique for each object.

The size of the hashtable also increases once it has reached the threshold of 0.75 and seems to use a formula of $6 \times 2^{n} - 1$. N being the amount of times the array has increased in length.

When the first input is stored, it is put at position 2, the centre of the table. Using my above equation, the object hash of “fred” is 3151467 which modulo 5, is equal to 2. This obviously shows that it’s equal to the position of the table that it was placed in, which proves my equation is correct.

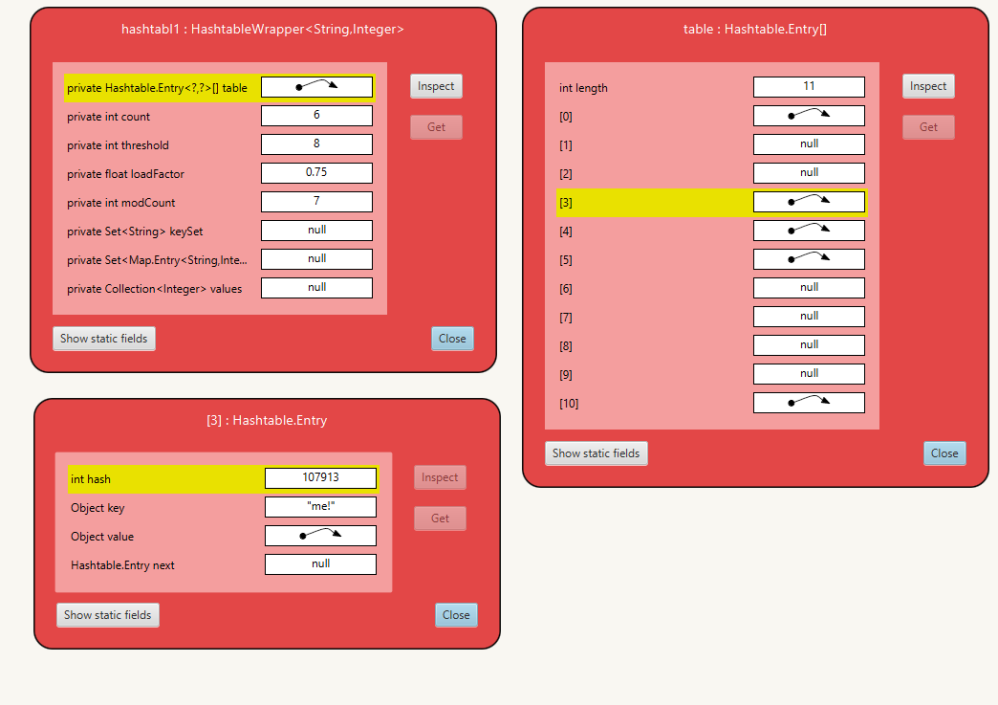


When I place the next equation, which has a hash of 3370, which modulo 5 is equal to 0, again the same as its position on the table.

When I get to placing the input of “but”, 999 the length of the table increases and the already placed objects are moved along with the new size of the table. I check the previous entries and change the equation from modulo 5, to modulo 11 the new length of the table. The results of this show that all the places of the elements have been moved to match the new length of the table. The table increases as I described earlier as well.

The next thing I notice upon adding the “not” entry, is that “is” is overwritten by this entry because the equation of its placement is equal to 4, the same as “is”. Rather than moving the entries about in any manner the previous entry is overwritten. Another possible idea is that the program would have bumped it up one spot but the entry in the fifth position is still “dead” which leads me to believe that it didn’t try to move the entry one spot and in fact this value was just overwritten.

The final value of “me!” just follows normal behaviour and does the formula for findings its position using the length of 11 and places it at position three.



Week 10

```
public class DepthFirstTraversal <T> extends AdjacencyGraph <T> implements
Traversal <T> {
	private List<T> traversal = new ArrayList<T>();
	private List<T> visited = new ArrayList<T>();

	@Override
	public List<T> traverse() throws GraphError {
		while(visited.size() < getNodes().size()) { //Makes sure that the
visited array is smaller than the amount of nodes
			for (T node: getNodes()) { //Goes through the nodes one by one
to build the array
				getUnvisitedNode(node); //Checks the node to see if it
has been visited before
				traverse(node); //Beings the depth first traversal with
the new node
			}
		}
		return traversal; // Returns the array once its completed
	}

	void traverse(T node) throws GraphError {
		for (T neighbour: getNeighbours(node)) { //Goes through each
neighbour of the node
			node = neighbour; //Changes the node to the neighbour
			if(node != null && !visited.contains(node)) // If the node
hasn't been visited(to prevent going to the same neighbour over and over)
			{
				traversal.add(node); //Adds the node to the array
				visited.add(node); //Adds the node to the visited array
so it prevents infinite loops
				traverse(node); //Traverses again
			}
		}
	}

	T getUnvisitedNode(T node) throws GraphError {
		visited.add(node); //Adds the node to the visited array to prevent
infinite loops
		return node; //Returns the node to the traverse
	}
```

Week 11

```
public class RefCountTopologicalSort<T> extends AdjacencyGraph<T> implements
TopologicalSort<T> {
	private HashMap<T,Integer> refCountTable = new HashMap<T,Integer>();
	private Stack<T> sort = new Stack<T>();

	@Override
	public List<T> getSort() throws GraphError {
		setUpRefCounts();
		sort();

		return sort;
	}

	private void setUpRefCounts() throws GraphError {
		initialiseRefCounts();
		countReferences();
	}

	private void countReferences() throws GraphError {
		for (T node: getNodes()) { //Get all the nodes and go through them
one by one
			for (T neighbour: getNeighbours(node)) { //Get the neighbours
of the selected node
				int currentCount = refCountTable.get(neighbour); //Get
the number of children of the neighbour
				refCountTable.put(neighbour, ++currentCount);
//Increment the current count in the refCountTable for the amount of neighbours
			}
		}
	}

	private void initialiseRefCounts() {
		for (T node: getNodes()) { // Gather the nodes
			refCountTable.put(node, 0); // Declare all nodes in the table
as a count of 0
		}
	}

	private void sort() throws GraphError {
		T node; // Declare the node as a variable

		while ((node = nextReferenceZeroNode()) != null) { // Check the
nodes next reference  and continue the loop whilst it isn't null
			for (T neighbour: getNeighbours(node)) { // Check the
neighbours of the node
				Integer count = refCountTable.get(neighbour); // Get
the amount of neighbours from the refCountTable
				if (count != null) {
					refCountTable.put(neighbour, count-1);  // Reduce
the count from the refCountTable by 1 for each neighbour
				}
				refCountTable.put(node, count-1);
			}
		refCountTable.remove(node); // Remove the node from the
refCountTable
		sort.add(node); // Add the node to the sort
		}
```

```
        }

    private T nextReferenceZeroNode(){
        for (Entry<T, Integer> entry : refCountTable.entrySet()) { // Get each T
and Integer from the entrySet
            if(entry.getValue() == 0){
                return (T) entry.getKey(); // Returns the getKey to the sort
            }
        }
        return null;
    }


       /**
        * Runs the sort in RefCountTopologicalSort and checks to see
        * if the result equals the result used in the Lecture Notes.
        * @throws GraphError
        */    @Test
       public void test() throws GraphError {
        RefCountTopologicalSort<Integer> graph =new
RefCountTopologicalSort<Integer>();
        Integer node0 = new Integer(0);
        Integer node1 = new Integer(1);
        Integer node2 = new Integer(2);
        Integer node3 = new Integer(3);
        Integer node4 = new Integer(4);
        Integer node5 = new Integer(5);
        Integer node6 = new Integer(6);
        Integer node7 = new Integer(7);
        Integer node8 = new Integer(8);
        Integer node9 = new Integer(9);
        graph.add(node0);
        graph.add(node1);
        graph.add(node2);
        graph.add(node3);
        graph.add(node4);
        graph.add(node5);
        graph.add(node6);
        graph.add(node7);
        graph.add(node8);
        graph.add(node9);
        graph.add(1, 5);
        graph.add(0, 5);
        graph.add(1, 7);
        graph.add(3, 2);
        graph.add(3, 4);
        graph.add(3, 8);
        graph.add(6, 0);
        graph.add(6, 1);
        graph.add(6, 2);
        graph.add(8, 4);
        graph.add(8, 7);
        graph.add(9, 4);

        assertEquals(new String("[3, 6, 0, 1, 2, 5, 8, 7, 9, 4]"),
Arrays.toString(graph.getSort().toArray()));
       }
```

Week	Overall	Documentation	Structure	Names	Tests	Function
8 Hashtable	A					
10 Depth First Traversal	C	B	B	B	D	A
11 Reference Counting Topological Sort	A	A	A	B	B	A

Week 8:

I think for this week I deserve a grade A because I believe I clearly displayed and showed an understanding of hashtables. I talked about the memory issues when an element is placed in the same location, how it can be placed in the same location and screenshots to show each step and my explanation of the step.

Week 10:

This week I believe I deserve a grade C due to the fact I fell weak on testing. I believe I deserve a grade A on function since the code does actually function correctly when tested with a System.out and since it is commented throughout and the structure and names are good I believe this week deserves a C despite being dragged down by testing.

Week 11:

I believe this week deserves an overall A due to the fact I think my code in this week is strong in most aspects. I believe my documentation is strong since it is commented throughout and the structure of the code is good. I also believe the naming is good since all method names and variables are clear to understand by their names alone. I believe my testing is good since it does what is required, by making sure the correct sort is returned as was described in the lecture notes. I believe the function is good for the same reason.

Week 13

Judging by the tests I ran it does always seem to terminate after a finite amount of time, usually between 30 seconds to a couple of minutes. Due to this I am confident in saying that it will always terminate if left running long enough.

The shortest possible outcome I received was 14 lines. Since two lines were outputting the “has finished” outputs and it was also counting the starting lines at 0, this is the shortest possible outcome if it counts straight from 10 to 0 and vice versa.

The largest possible outcome is 11 since if it reaches 10 it will automatically count up one and then will move down and hit 10, finishing this counter.

The smallest value the counter that can be reached is -1, this happens when it starts at 0 and counts down one. It will soon correct itself and finish this counter since it will attempt to move towards the finishing value.

Week 14

The problems with the first scenario are that it could end up with a loop only allowing the Bolivians to use the pass since as soon as they leave and remove the stone they could try to enter again as soon as they've left since the Peruvians could be resting.

If they both check for stones at the same time and place a stone at the same time they could both end up in the tunnel at the same time, causing a crash.

After making the changes and leaving the program to time out a couple of times I didn't encounter any issues so I do believe that this method prevents crashes between the trains.

I'm not sure why there was a dispute over timetabling since both trainlines get an equal amount of passes and take turns in doing so.

Peru

```
    /**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more
than one train in the
     * pass at any one time.
     */
      public void runTrain() throws RailwaySystemError {
      Clock clock = getRailwaySystem().getClock();
      Basket basket = getBasket();
      Railway nextRailway = getRailwaySystem().getNextRailway(this);
      while (!clock.timeOut()) { // Whilst the clock is ticking
            choochoo();
            basket.putStone(this); // Place a stone in this railway systems
basket
            while (nextRailway.getBasket().hasStone(this)) { // While the next
railway system has a stone in its basket
                  if(!getSharedBasket().hasStone(this)) { // If this has stone
or doesnt or the other railway has stone or doesnt
                        basket.takeStone(this);
                        while(!getSharedBasket().hasStone(this)) { // While the
next railways basket is not equal to this railways basket
                              siesta();
                        }
                        basket.putStone(this);
                  }
            }
            crossPass();
            basket.takeStone(this);
            getSharedBasket().takeStone(this);
      }
    }
```

Bolivia

```
/**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more
than one train in the
     * pass at any one time.
     */
    public void runTrain() throws RailwaySystemError {
      Clock clock = getRailwaySystem().getClock();
      Basket basket = getBasket();
      Railway nextRailway = getRailwaySystem().getNextRailway(this);
      while (!clock.timeOut()) { // Whilst the clock is ticking
            choochoo();
            basket.putStone(this); // Place a stone in this railway systems
basket
            while (nextRailway.getBasket().hasStone(this)) { // While the next
railway system has a stone in its basket
                  if(getSharedBasket().hasStone(this)) { // If the shared basket
has a stone
                        basket.takeStone(this);
                        while(getSharedBasket().hasStone(this)) { // While the
shared basket has the stone
                              siesta();
                        }
                        basket.putStone(this);
                  }
            }
            crossPass();
            basket.takeStone(this);
            getSharedBasket().putStone(this);
      }
    }
```

Week 15

```
System will run for 20.0s
System will start with producer taking up to 1.0s for each buffer access
and consumer taking up to 1.0s
System will then change to producer taking up to 1.0s
and consumer taking up to 1.0s for each buffer access
Consumer: Retrieving data item
Producer: adding 0
```

When the criticalSection.poll and noOfElements.poll are swapped it causes the buffer to freeze. This is due to the fact that there is no permit in the semaphore which causes this thread to stop and wait until it does have a permit, which will never happen and the it will timeout and move to the next test.

In the table below I've shown where the process freezes, when criticalSection.poll see's there are no spaces in the criticalSection and the program freezes there.

Put	Get	noOfSpaces	noOfElements	criticalSection
		10	0	1
	criticalSection.poll	10	0	0
noOfSpaces.poll		9		

Put	Get	noOfSpaces	noOfElements	criticalSection
		10	0	1
	noOfElements.poll	10	0	1
criticalSection.poll		9		

When noOfSpaces.poll and criticalSection.poll is swapped in the put method it is immediately apparent that the order is not essential in this method and criticalSection still has a permit allowing the semaphore to run normally.

Week 16

```
public LockResourceManager(Resource resource, int maxUseages) {
		super(resource, maxUseages);

		// Define the locks for each possible priority
	 for (int priority = 0; priority < NO_OF_PRIORITIES; priority++) {
		conditions[priority] = lock.newCondition();
	 }
	}
	// Request resource method ran by the resource user class
	public void requestResource(int priority) throws ResourceError {
		// Locks the method to one process
		lock.lock();

		try {
			if (resourceInUse) {
				// Increases the number of processes waiting for this
priority
				increaseNumberWaiting(priority);
				conditions[priority].await();
			}
		// Sets the resource in use boolean to true until set to false once
the resource has been used
		resourceInUse = true;
		} catch (InterruptedException e) {
			e.printStackTrace();
		} finally {
			lock.unlock();
		}
	}
	public int releaseResource() throws ResourceError {
		int highestPriority = NONE_WAITING;
		lock.lock();
		try {
			// For loop which goes through each priority to find the
highest priority waiting
			for (int i = 0; i < NO_OF_PRIORITIES; i++) {
				if(getNumberWaiting(i) > 0) {
					highestPriority = i;
				}
			}
			resourceInUse = false;
			if (highestPriority == -1) {
				return NONE_WAITING;
			} else {
				decreaseNumberWaiting(highestPriority); // Decreases
the number of processes waiting with the highest priority
				conditions[highestPriority].signal();  // Signals the
process with the highest priority waiting to run
				return highestPriority;
			}
		} finally {
			lock.unlock();
		}
	}
```

My code as somewhat described in the comments starts by building the conditions for each possible priority so that they can be enabled and disabled later when needed. Then the request resource is ran in the codes main() and locks the method so only one thread can access it and checks if the resourceInUse bool is enabled and if not goes past and enables it, then the increaseNumberWaiting method is ran and increments the value for the threads current priority and begins to await whilst the current priority is signalled in the releaseResource method and after the method is unlocked.

The releaseResource method starts by initialising the integer and assigning it the NONE_WAITING value and then the thread is locked in. It then runs the for loop until it gets to the highest waiting priority using the getNumberWaiting method. The resourceInUse is set to false and if the highest priority is -1 as assigned earlier, it returns the NONE_WAITING value and the task is complete. Otherwise it goes on to run the decreaseNumberWaiting and then signal the appropriate condition. Then the method returns the next highest waiting priority. This loops around until the resource is exhausted.

```
Starting Process "2" (priority: 0)
Starting Process "1" (priority: 0)
Starting Process "3" (priority: 0)
Starting Process "4" (priority: 0)
Process "2" (priority: 6) is requesting resource "A"
Process "1" (priority: 5) is requesting resource "A"
Process "3" (priority: 2) is requesting resource "A"
Process "2" (priority: 6) gained access to resource "A"
Process "4" (priority: 6) is requesting resource "A"
2 is using resource "A"
2 has finished using resource "A"
resource "A" has 2 uses left
Process "2" (priority: 6) released resource "A", to a process with priority 6
Process "4" (priority: 6) gained access to resource "A"
4 is using resource "A"
Process "2" (priority: 8) is requesting resource "A"
4 has finished using resource "A"
resource "A" has 1 uses left
Process "4" (priority: 6) released resource "A", to a process with priority 8
Process "2" (priority: 8) gained access to resource "A"
2 is using resource "A"
2 has finished using resource "A"
resource "A" has 0 uses left
Process "2" (priority: 8) released resource "A", to a process with priority 5
Process "1" (priority: 5) gained access to resource "A"
Process "1" (priority: 5) cannot use resource "A" as the resource is exhausted
resource "A" has 0 uses left
Process "1" (priority: 5) released resource "A", to a process with priority 2
Process "3" (priority: 2) gained access to resource "A"
Process "3" (priority: 2) cannot use resource "A" as the resource is exhausted
resource "A" has 0 uses left
Process "3" (priority: 2) released resource "A", there were no waiting processes
Process "2" (priority: 8) has finished
Process "1" (priority: 5) has finished
Process "3" (priority: 2) has finished
Process "4" (priority: 6) has finished
All processes finished
```

Week	Overall	Documentation	Structure	Names	Tests	Function
13 Counter Behaviour	**B**					
14 Dekker Trains	**A**	**A**	**A**	**A**	**A**	**A**
15 Semaphore Behaviour	**C**					
16 Locks and Conditions	**A**	**A**	**B**	**A**	**A**	**A**

Week 13:

I believe for week 13 I deserve a grade B since I did show evidence of running the code multiple times by describing my understanding of the minimum and maximum values.

Week 14:

I believe I deserve a grade A for this week since most aspects of this week are strong. I believe I deserve an A for documentation since I did comment throughout the code and also write about my understanding in the logbook. I believe the structure couldn't be improved since it follows the description in the lecture notes. The names I also don't believe could be improved since most are given already and likewise for the tests since the test is built in and running it shows the code functions. The functionality I also believe deserves a grade A since it does function when tested.

Week 15:

I believe I deserve a B for this week since I showed an understanding of the code and detailed how it functions in the table. I detailed how the code behaved when swapping the critsec and noofspaces and described my understanding of it in the logbook.

Week 16:

For this week I believe I deserve I deserve a grade A since I believe my documentation and testing is good. I included a console log which shows the code functioning as it should which with my code commenting I believe shows good documentation. For these reasons I also believe the testing and functionality deserves an A.

Week 17

```
NOT = [
0 1
1 0];

AND = [1,1,1,0;0,0,0,1];


%Tensor Product NOT AND
TPNANDOR = [
     0     0     0     0     1     1     1     0
     0     0     0     0     0     0     0     1
     1     1     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
];

%Logbook Answer
Answer = [
     0     0     0     0     1     1     1     0
     1     1     1     1     0     0     0     1
];
```

Week 20

```
% Hadamard * ZERO = 0.7071
ans =

    0.7071
    0.7071

% Hadamard * 0.7071 = 1 or 0
ans =

    1.0000
   -0.0000

% The results with ONE are the same

% Hadamard * ONE = 0.7071
ans =

    0.7071
   -0.7071

% Hadamard * ans =
ans =

  -0.0000
   1.0000
```

In the first instance A is either one or zero, B is always 0.7071 and C is either one or zero. As shown below:-

A = $\begin{bmatrix}1\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\end{bmatrix}$

B = $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0.7071\\0.7071\end{bmatrix}$

B = $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0.7071\\-0.7071\end{bmatrix}$

C = $\begin{bmatrix}1\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\end{bmatrix}$

This shows that after passing through the Hadamard gate at the instance of C the value returns to the initial value.

Week	Grade	Criteria
17 Modelling Circuits	**C**	I've derived a matrix for the half-adder and correctly and applied the matrix methods for constructing sequential and parallel circuits and justified it with the math in matlab but did not test it with various inputs. I would still give this week a grade C due to the fact I showed a generally correct matrix model.
20 Quantum Computing	**B**	I fully analysed the values at A, B and C and discussed the relationship between the values at A and C. I didn't discuss what the implications of a purely probabilistic model would be for maintaining this relationship. I did realise the input returns the initial value and explain it in my logbook however and my working for the matrix arithmetic however and I believe I deserve a B for this however.

General Self-Assessment

Assessment Criterion	Grade
Answers to flagged logbook questions	A
Answers to other practical questions	C
Other practical work (additional exercises you have undertaken of your own initiative)	F
Understanding of the module material to date	A
Level of self-reflection and evaluation	A
Participation of timetabled activities	A
Time spent outside timetabled classes (guideline is two hours self-study for each hour of timetabled study)	A

Answers to flagged logbook questions:

I would grade myself an A for this since I finished every flagged logbook question from each week and have shown it in the logbook.

Answers to other practical questions:

I would grade myself a C for this since I attempted some other questions from the tutorial exercises and believe I have shown my answers to a few in the logbook.

Other practical work:

I would grade myself a F for this since I don't think I built on any of the answers entirely on my own initiative.

Understanding of the module material to date:

I would grade myself an A on this since I after answering each logbook question it greatly improved my understanding of the topic.

Level of self-reflection and evaluation:

I would grade myself an A for this since I believe I gave a strong self-reflection within the logbook.

Participation of timetabled activities:

I would grade myself an A for this since I believe I attended every tutorial and stayed for extra ones and participated in the activities of the tutorials.

Time spent outside of timetabled classes:

I would grade myself an A for this since I spent a great deal more of the time outside of tutorials working on my answers.