

Algorithms, Processes and Data

Logbook

Jack Hurst

Week 1 & 2

a)

```
protected void randomise()
{
    //start timer
    setUp();

    //create an int array and fill it

    int[] copy = getArray();

    //create int j

    int j;

    /*
     * this for loop will swap position i and a random position
     */

    //for every int in copy
    for(int i : copy)
    {
        //store a random int in the array

        int randomPos = getRandomIndex();

        //make j equal pos i

        j = copy[i];

        //make position i of the array equal the one from a random pos

        copy[i] = copy[randomPos];

        //make a randompos in the array equal the the one in pos i

        copy[randomPos] = j;

    }

    //set the array to equal the new randomised copy

    for (int index = 0; index < getArray().length; index ++)
    {
        getArray()[index] = copy[index];
    }
    //end timer
    tearDown();
}
```

b)

```

//get time at the start
protected void setUp()
{
    testStart = System.nanoTime();
}
//get time at the end
protected void tearDown()
{
    testEnd = System.nanoTime();
}
//return the the time in microseconds
protected String printTime()
{
    String time = "\n" + "Test " + "took " + (testEnd-testStart)/1000 + "
microseconds";
    return time;
}

```

Tests:

```

public class CleverRandomListingTest extends ListingTest
{
    private static long testStart, testEnd;

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Rule public TestName testName = new TestName();

    @Before
    public void setUp() throws Exception {
        testStart = System.nanoTime();
    }

    @After
    public void tearDown() throws Exception {
        testEnd = System.nanoTime();
        System.out.println("Test \"" + testName.getMethodName() + "\"
took " + (testEnd-testStart)/1000 + " microseconds");
    }

    @Test
    public void testOneSize() {
        testSize(1, new CleverRandomListing(1));
    }

    @Test
    public void testOneContents() {
        testContents(1, new CleverRandomListing(1));
    }
}

```

```

@Test
    public void testTwoSize() {
        testSize(2,new CleverRandomListing(2));
    }

    @Test
    public void testTwoContents() {
        testContents(2,new CleverRandomListing(2));
    }
    @Test
    public void testFourSize() {
        testSize(4,new CleverRandomListing(4));
    }

    @Test
    public void testFourContents() {
        testContents(4,new CleverRandomListing(4));
    }
    @Test
    public void testHundredSize() {
        testSize(100,new CleverRandomListing(100));
    }

    @Test
    public void testHundredContents() {
        testContents(100,new CleverRandomListing(100));
    }
    @Test
    public void testThousandContents() {
        testContents(1000,new CleverRandomListing(1000));
    }

    @Test
    public void testMillionSize() {
        testSize(1000000,new CleverRandomListing(1000000));
    }
}

```

```

Test "testHundredSize" took 37993 microseconds
Test "testOneSize" took 34 microseconds
Test "testHundredContents" took 720 microseconds
Test "testTwoSize" took 27 microseconds
Test "testOneContents" took 26 microseconds
Test "testTwoContents" took 37 microseconds
Test "testThousandContents" took 15724 microseconds
Test "testMillionSize" took 128112 microseconds
Test "testFourSize" took 101 microseconds
Test "testFourContents" took 20 microseconds

```

Week 3 & 4

```
public static <T> boolean equals(T object1,T object2)
{
    if (object1==null)
    {
        return object2==null;
    }
    else
    {
        return object1.equals(object2);
    }
}
public static <T> void Swap(T[] array,int i, int j)
{
    //Sets two variables to equal the positions I wants to swap
    T firstObject = array[i];
    T secondObject = array[j];
    //Set both positions to equal the stored variables
    array[i] = secondObject;
    array[j] = firstObject;
}
public static void main(String[] args)
{
    //Makes the array which has objects that need swapping
    Object[] names = {1,"Andrew",2,"Diane",3,"Simon"};
    //Runs the swap method
    Swap(names, 1, 4);
    //Prints to the debug log for testing purposes
    System.out.println(Arrays.toString(names));
}
```

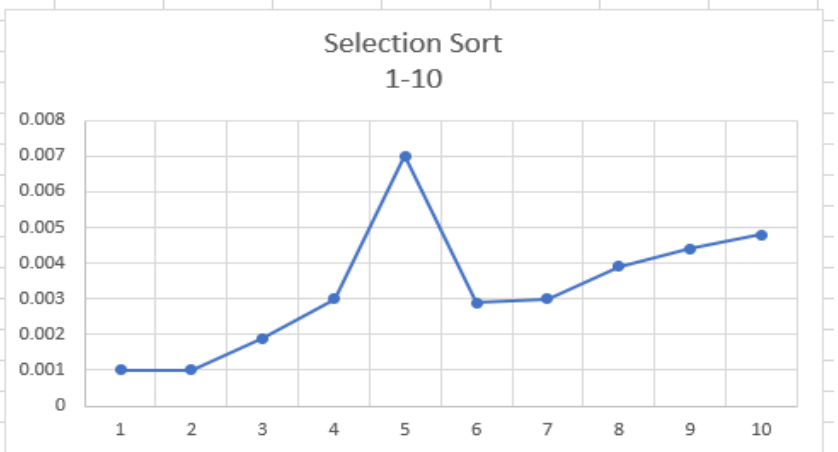
Results:

[1, 3, 2, Diane, Andrew, Simon]

Week 5

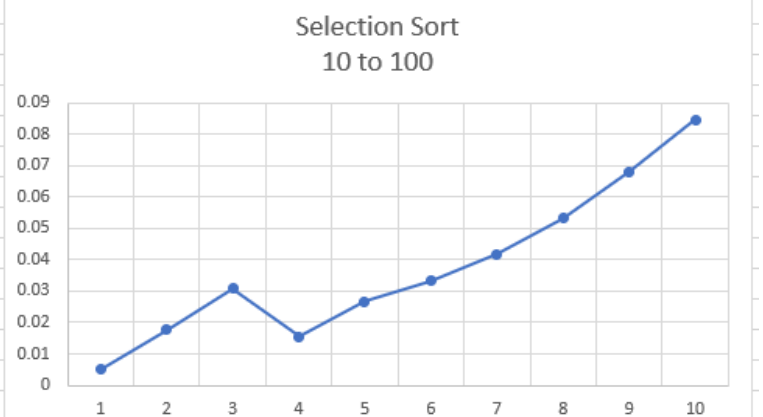
Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
1	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
2	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
3	0.002	0.001	0.001	0.001	0.004	0.001	0.001	0.004	0.001	0.003	0.0019
4	0.002	0.003	0.002	0.002	0.003	0.004	0.002	0.002	0.005	0.005	0.003
5	0.005	0.007	0.005	0.014	0.007	0.005	0.004	0.007	0.009	0.007	0.007
6	0.004	0.003	0.005	0.002	0.003	0.003	0.002	0.003	0.002	0.002	0.0029
7	0.003	0.002	0.004	0.003	0.004	0.002	0.002	0.002	0.003	0.005	0.003
8	0.003	0.003	0.003	0.003	0.003	0.004	0.004	0.004	0.008	0.004	0.0039
9	0.003	0.003	0.003	0.003	0.004	0.006	0.005	0.006	0.007	0.004	0.0044
10	0.004	0.004	0.004	0.004	0.005	0.007	0.004	0.005	0.006	0.005	0.0048

It is difficult to determine the relationship between the increase of size and the time taken as the size is so small. This could contribute to the anomaly at size 5. See next sheet for size 10 - 100.



Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
10	0.005	0.01	0.004	0.004	0.004	0.005	0.004	0.004	0.004	0.006	0.005
20	0.018	0.023	0.015	0.021	0.017	0.019	0.015	0.016	0.017	0.015	0.0176
30	0.031	0.025	0.028	0.034	0.029	0.031	0.031	0.035	0.03	0.035	0.0309
40	0.014	0.02	0.015	0.014	0.015	0.015	0.015	0.017	0.015	0.015	0.0155
50	0.026	0.032	0.026	0.027	0.023	0.024	0.031	0.025	0.024	0.029	0.0267
60	0.033	0.04	0.033	0.033	0.032	0.032	0.028	0.032	0.036	0.034	0.0333
70	0.043	0.041	0.047	0.039	0.041	0.039	0.038	0.045	0.043	0.043	0.0419
80	0.055	0.056	0.055	0.049	0.052	0.052	0.052	0.054	0.055	0.053	0.0533
90	0.068	0.068	0.064	0.06	0.066	0.066	0.063	0.07	0.087	0.069	0.0681
100	0.087	0.09	0.083	0.081	0.085	0.08	0.084	0.085	0.087	0.085	0.0847

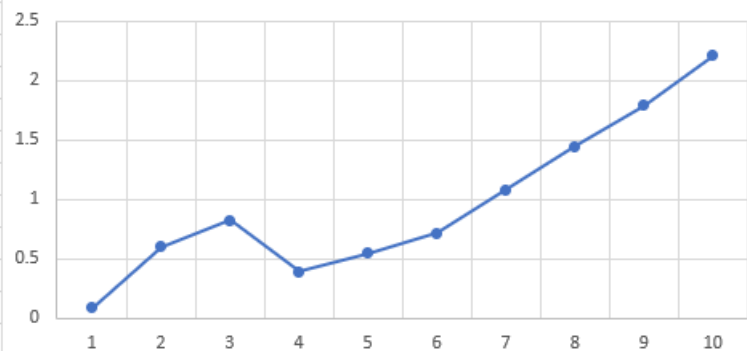
See previous sheet for size 1 - 10. The correlation between size and time taken is clearer, however these are still small sizes and there is an anomaly at size 3 however it is less severe. See next sheet for size 100 - 1000.



Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
100	0.08	0.086	0.087	0.078	0.08	0.078	0.097	0.082	0.077	0.087	0.0832
200	0.648	0.665	0.661	0.341	0.734	0.328	0.717	0.592	0.71	0.63	0.6026
300	0.0889	0.875	0.524	1.126	0.931	1.137	1.131	0.955	0.933	0.547	0.82479
400	0.34	0.337	0.389	0.319	0.444	0.306	0.534	0.418	0.438	0.39	0.3915
500	0.512	0.489	0.598	0.473	0.616	0.493	0.584	0.608	0.581	0.476	0.543
600	0.0769	0.686	0.84	0.7	0.881	0.708	0.749	0.904	0.921	0.697	0.71629
700	0.997	0.995	1.118	0.983	1.15	0.956	1.185	1.213	1.224	0.971	1.0792
800	1.374	1.362	1.611	1.331	1.646	1.413	1.381	1.5	1.508	1.307	1.4433
900	1.654	1.668	1.943	1.583	2.03	1.735	1.808	1.932	1.944	1.592	1.7889
1000	2.141	2.024	2.558	1.935	2.251	2.081	2.107	2.476	2.508	1.992	2.2073

See previous sheet for size 10 - 100. Similar analysis to the previous sheet, has an anomaly at size 3. See next sheet for size 1000 - 10000.

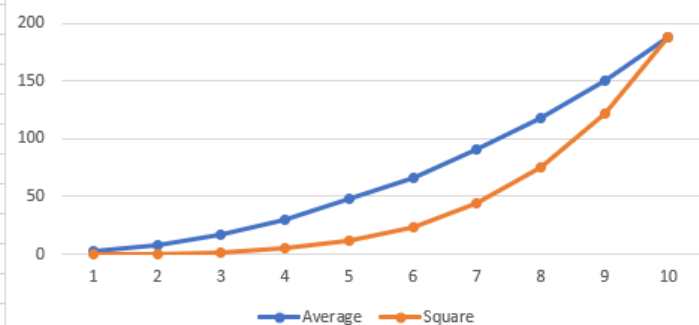
Selection Sort
100 - 1000



Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average	Square
1000	1.89	1.978	1.93	2.068	2.395	1.843	1.805	1.951	2.506	2.039	2.0405	0.020405
2000	7.641	7.588	7.961	8	8.694	7.809	7.528	7.592	9.039	7.375	7.8949	0.315796
3000	17.553	17.056	16.689	16.789	18.653	17.163	16.735	16.59	19.302	17.301	17.3831	1.564479
4000	29.25	28.9	30.143	30.138	33.184	31.105	29.515	28.715	34.605	29.565	30.512	4.88192
5000	47.039	44.922	47.643	48.175	49.913	47.684	45.433	46.454	52.313	45.561	47.5137	11.87843
6000	66.222	65.946	65.502	67.629	71.178	66.037	64.559	64.727	71.167	65.171	66.8138	24.05297
7000	88.766	88.025	90.366	89.648	97.403	88.75	89.129	88.53	96.546	89.743	90.6906	44.43839
8000	115.771	115.075	117.177	117.09	126.69	116	115.171	115.978	128.175	117.006	118.3895	75.76928
9000	147.169	146.498	149.481	149.162	158.932	146.747	146.656	146.142	159.331	150.127	150.0245	121.5198
10000	183.024	184.702	185.174	185.023	197.428	186.132	183.636	185.32	200.631	186.615	187.7685	187.7685

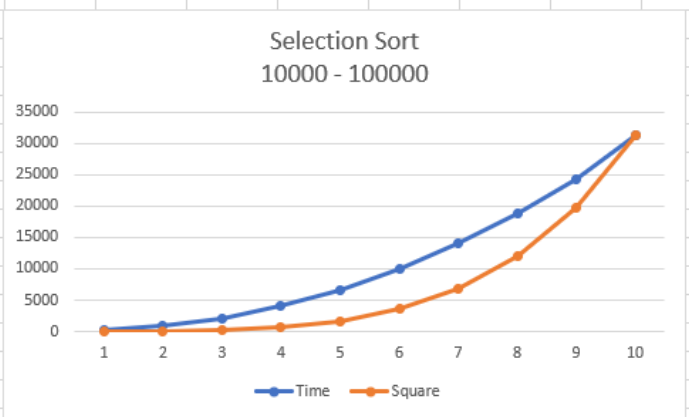
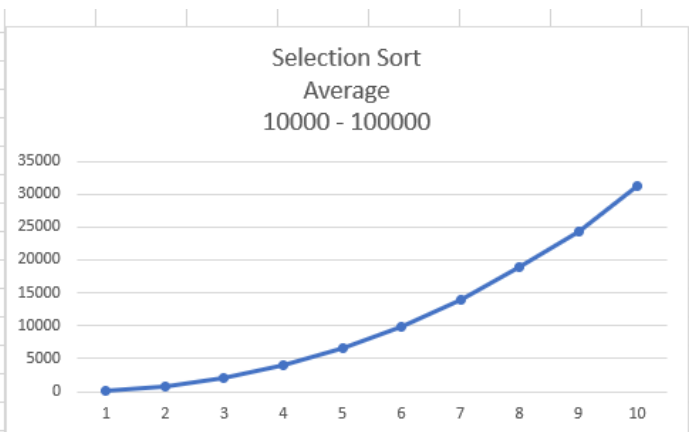
See previous sheet for size 100-1000. There is now a much smoother curve with not significantly noticeable anomalies. There is an extra plot showing the average value for the arrays at size 1000 multiplied by the square of difference in size of the arrays. There is an obvious similarity between the two lines albeit the average is a bit faster as the sizes get bigger. See next sheet for size 10000 - 100000

Selection Sort
1000-10000



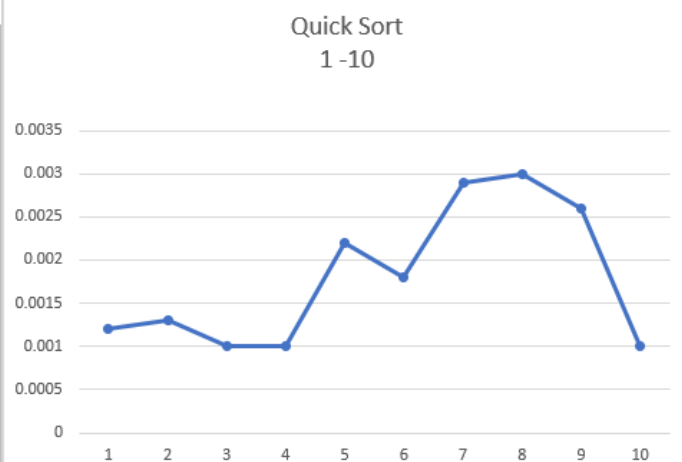
Size	Time	Square
10000	195.721	1.95721
20000	841.242	33.64968
30000	2099.53	188.9577
40000	4021.208	643.39328
50000	6613.808	1653.452
60000	9949.696	3581.8906
70000	13997.6	6858.826
80000	18812.39	12039.93
90000	24380.14	19747.911
100000	31260.3	31260.297

See previous sheet for size 1000- 10000. As can be seen in these graphs the increase in average is surprisingly smooth for one set of tests however could be smoother. It does increase much faster than the square plot.



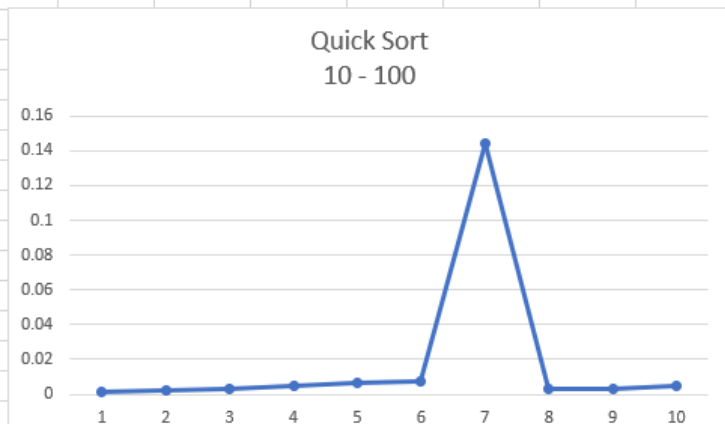
Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
1	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.001	0.001	0.0012
2	0.001	0.001	0.003	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.0013
3	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
4	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
5	0.001	0.001	0.002	0.001	0.003	0.003	0.001	0.001	0.004	0.005	0.0022
6	0.001	0.001	0.002	0.003	0.001	0.003	0.002	0.002	0.002	0.001	0.0018
7	0.006	0.002	0.003	0.003	0.002	0.001	0.001	0.005	0.002	0.004	0.0029
8	0.002	0.004	0.001	0.002	0.002	0.001	0.001	0.013	0.002	0.002	0.003
9	0.002	0.001	0.001	0.002	0.001	0.002	0.001	0.003	0.003	0.01	0.0026
10	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

The relationship between times and size are not clear, this is probably due to the small sizes being used. See next sheet for size 10 - 100.



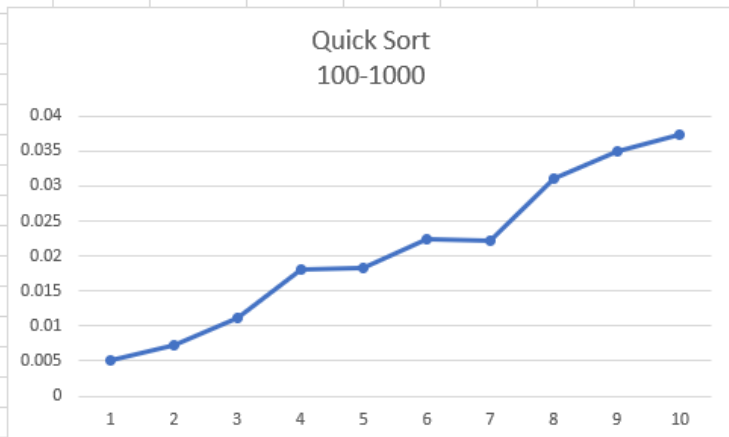
Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
10	0.001	0.001	0.001	0.003	0.001	0.001	0.001	0.001	0.001	0.001	0.0012
20	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.003	0.002	0.002	0.0021
30	0.002	0.004	0.003	0.002	0.003	0.003	0.002	0.005	0.002	0.002	0.0028
40	0.006	0.006	0.004	0.006	0.004	0.004	0.005	0.006	0.003	0.003	0.0047
50	0.005	0.007	0.011	0.009	0.007	0.005	0.005	0.004	0.005	0.004	0.0062
60	0.005	0.006	0.006	0.009	0.008	0.013	0.007	0.008	0.008	0.005	0.0075
70	0.142	0.14	0.141	0.138	0.138	0.146	0.164	0.146	0.151	0.14	0.1446
80	0.003	0.002	0.003	0.004	0.005	0.003	0.002	0.004	0.007	0.002	0.0035
90	0.003	0.003	0.004	0.004	0.002	0.002	0.003	0.003	0.003	0.003	0.003
100	0.005	0.009	0.003	0.003	0.004	0.003	0.005	0.004	0.005	0.003	0.0044

See previous sheet for size 1 - 10. If you do not take into account the very large anomaly then there may be a slight correlation between size and times. However the sizes are still too small to tell a great difference. See the next page for size 100 - 1000.



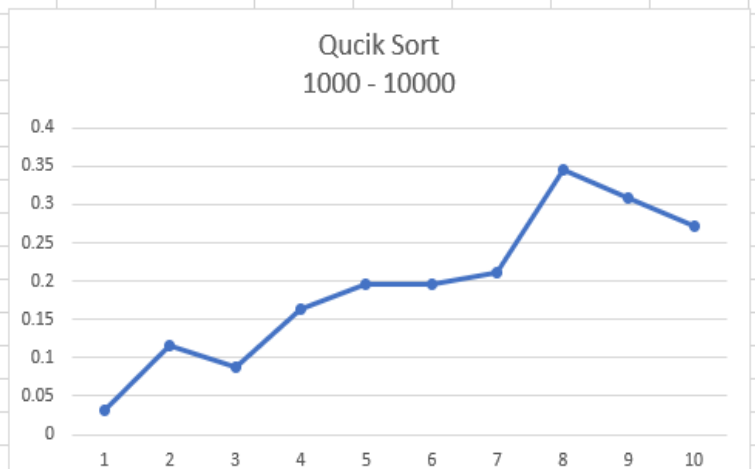
Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
100	0.008	0.006	0.008	0.003	0.005	0.003	0.003	0.008	0.003	0.004	0.0051
200	0.008	0.008	0.010	0.006	0.006	0.006	0.004	0.008	0.010	0.006	0.0072
300	0.012	0.012	0.013	0.011	0.008	0.013	0.011	0.008	0.016	0.008	0.0112
400	0.026	0.022	0.022	0.013	0.018	0.017	0.018	0.012	0.016	0.017	0.0181
500	0.027	0.016	0.024	0.014	0.021	0.015	0.016	0.019	0.014	0.017	0.0183
600	0.031	0.033	0.030	0.017	0.022	0.016	0.017	0.020	0.018	0.020	0.0224
700	0.023	0.042	0.022	0.018	0.019	0.015	0.022	0.022	0.019	0.019	0.0221
800	0.026	0.050	0.027	0.028	0.028	0.034	0.024	0.040	0.022	0.031	0.031
900	0.030	0.047	0.024	0.030	0.048	0.039	0.023	0.031	0.054	0.023	0.0349
1000	0.029	0.064	0.037	0.032	0.052	0.053	0.022	0.024	0.028	0.032	0.0373

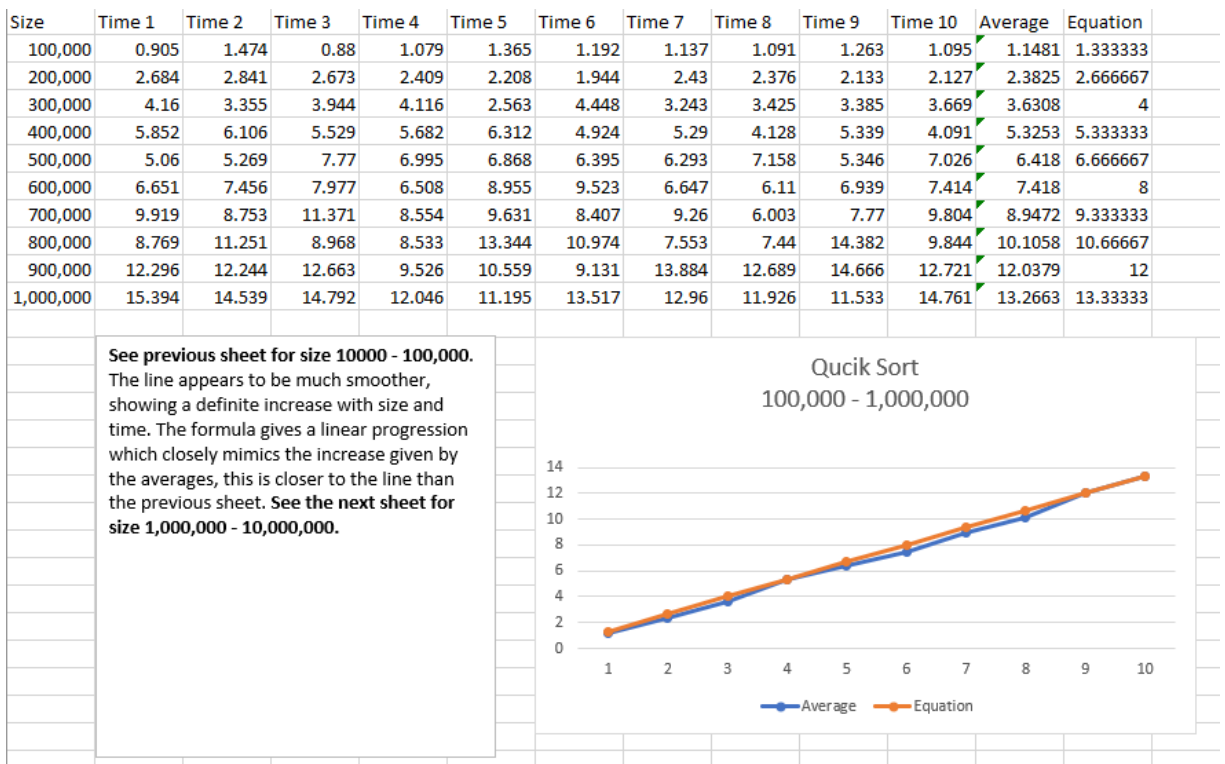
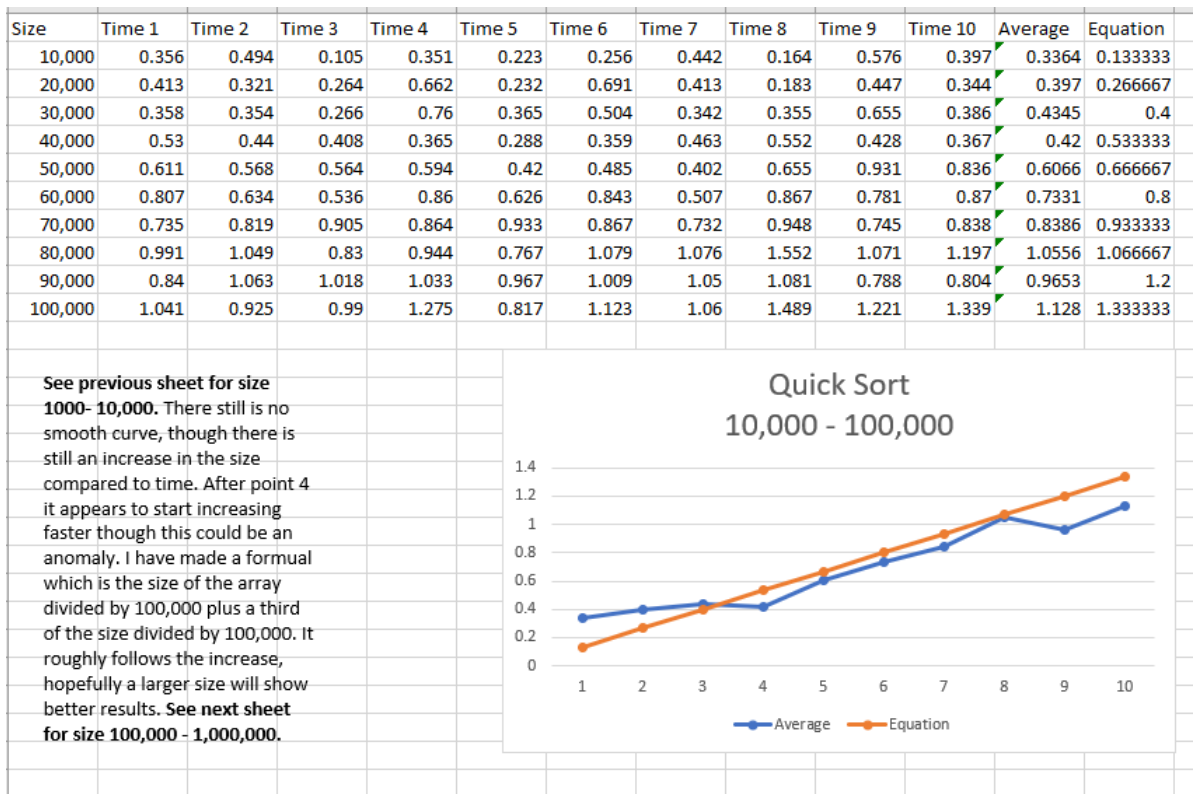
See the previous sheet for size 10 - 100. The relationship between size and time is a bit clearer and there isn't as a significant anomaly as on the previous sheet. However it still isn't very smooth, it may become clearer as the sizes increase. See the next sheet for size 1000 - 10000.

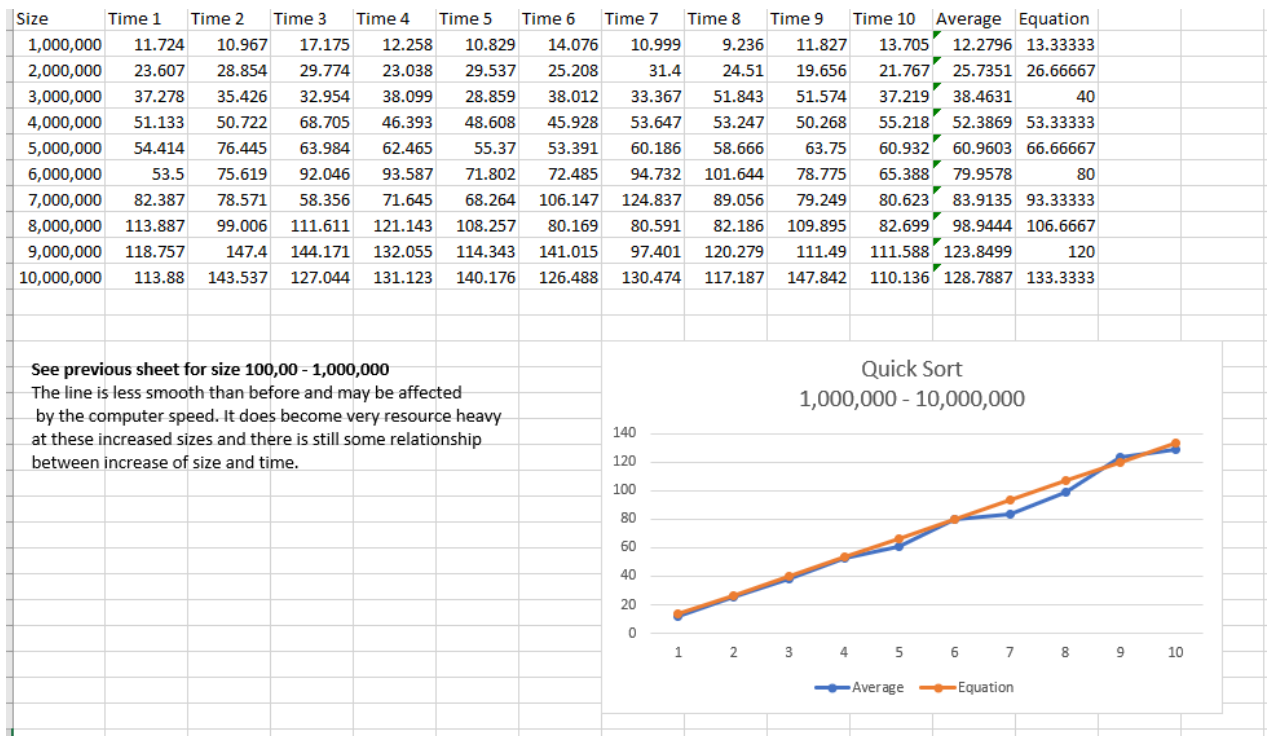


Size	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average
1000	0.026	0.02	0.025	0.02	0.028	0.035	0.037	0.041	0.051	0.03	0.0313
2000	0.063	0.043	0.05	0.056	0.044	0.064	0.065	0.64	0.076	0.057	0.1158
3000	0.065	0.08	0.098	0.088	0.095	0.08	0.097	0.096	0.107	0.072	0.0878
4000	0.109	0.097	0.128	0.256	0.176	0.261	0.189	0.12	0.175	0.116	0.1627
5000	0.196	0.275	0.174	0.203	0.145	0.266	0.152	0.136	0.217	0.206	0.197
6000	0.158	0.197	0.166	0.274	0.169	0.193	0.256	0.22	0.128	0.202	0.1963
7000	0.223	0.298	0.212	0.324	0.146	0.0363	0.184	0.288	0.21	0.201	0.21223
8000	0.0252	0.199	0.514	0.584	0.246	0.569	0.356	0.491	0.22	0.255	0.34592
9000	0.293	0.263	0.432	0.321	0.29	0.36	0.228	0.33	0.293	0.265	0.3075
10000	0.326	0.253	0.225	0.258	0.323	0.233	0.258	0.243	0.315	0.293	0.2727

See previous sheet for size 100 - 1000. The curve seems less clear compared to the previous sheet, there are a couple of major anomalies at size 2 and 8 and a slight one at 9. These make the relationship harder to determine. See next sheet for size 10,000 - 1,000,000







Week 6

1)

```
public void add(int index, T value) throws ListAccessError
{
    //temp node equals the value
    Node<T> temp = new Node<T>(value);

    //if it is empty
    if(isEmpty())
    {
        //head is the value
        head = temp;
        //tail equals what the head did
        tail = head;
    }
    else
    {
        //newNode will be the starting point, the head
        Node<T> newNode = head;
        int i = 1;
        //go through the list until you are at the node before the one
        //you want
        while(i < index)
        {
            newNode = newNode.getNext();
            i++;
        }
        //set the new node and the one before it
        temp.setNext(newNode.getNext());
        newNode.setNext(temp);
    }
    // add to the number of nodes
    noOfNodes ++;
}

@Override
public T remove(int index) throws ListAccessError
{
    Node<T> temp = head;
    //if its node empty
    if(!isEmpty())
    {
        int i = 1;
        //go through the list to the one before you want
        while(i != index)
        {
            temp = temp.getNext();
            i++;
        }
        //shift the nodes
        temp.setNext(temp.getNext().getNext());
        noOfNodes --;
    }
    return null;
}
```

```

@Override
    //get the value of the node at index
    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
    }

```

2)

```

@Test
public void randomArrayTestLow() throws ListAccessError
{

    long startTime = System.nanoTime();

    SinglyLinkedLists<Integer> list = new SinglyLinkedLists<Integer>();

    RandomIntegerArray rand = new RandomIntegerArray(500);

    Integer[] temp = rand.randomArray(500);

    for(int i = 0; i < temp.length; i++)
    {
        list.add(i, temp[i].intValue());
    }
    long endTime = System.nanoTime();

    System.out.println("Array Position: " + temp[375].toString() + " List Position: "
+ list.get(375) + " Time Taken in Microseconds: " + (endTime-startTime)/10000);

    assertEquals(temp[375],list.get(375));
}

```

Result:

Array Position: 174 List Position: 174 Time Taken in Microseconds: 61

```

@Test
public void randomArrayTestHigh() throws ListAccessError
{
    long startTime = System.nanoTime();

    SinglyLinkedLists<Integer> list = new SinglyLinkedLists<Integer>();

    RandomIntegerArray rand = new RandomIntegerArray(50000);

    Integer[] temp = rand.randomArray(50000);

    for(int i = 0; i < temp.length; i++)
    {
        list.add(i, temp[i].intValue());
    }

    long endTime = System.nanoTime();

    System.out.println("Array Position: " + temp[4756].toString() + " List Position: "
+ list.get(4756) + " Time Taken in Microseconds: " + (endTime-startTime)/10000);

    assertEquals(temp[4756],list.get(4756));

}

```

Result:

Array Position: 31283 List Position: 31283 Time Taken in Microseconds: 220797

Java - week06/src/LinkedList/SinglyLinkedListsTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 4.283 seconds

Runs: 11/11 Errors: 0 Failures: 0

LinkedList.SinglyLinkedListsTest [Runner: JUnit 4] (4.259 s)

- testGetNegative (0.045 s)
- testGetOutOfBounds (0.001 s)
- randomArrayTestHigh (4.197 s)
- testGet (0.001 s)
- testGetSingleton (0.000 s)
- testGetHead (0.000 s)
- testGetTail (0.000 s)
- randomArrayTestLow (0.002 s)
- testGetSingletonOutOfBounds (0.000 s)
- testGetSingletonNegative (0.001 s)
- testGetFromEmpty (0.001 s)

```

1 package linkedList;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7
8
9
10
11 public class SinglyLinkedListsTest
12 {
13     @Rule
14     public ExpectedException thrown = ExpectedException.none();
15
16     @Test
17     public void testGetFromEmpty() throws ListAccessError {
18         SinglyLinkedLists<Integer> list = new SinglyLinkedLists<Integer>();
19         thrown.expect(ListAccessError.class);
20         thrown.expectMessage("Index out of bounds");
21         list.get(0);
22     }
23
24     @Test
25     public void testGetSingleton() throws ListAccessError {
26         SinglyLinkedLists<Integer> list = new SinglyLinkedLists<Integer>();
27         list.add(0, 5);
28         assertEquals(new Integer(5),list.get(0));
29     }
30
31     @Test
32     public void testGetSingletonNegative() throws ListAccessError {
33         SinglyLinkedLists<Integer> list = new SinglyLinkedLists<Integer>();
34         list.add(0, 5);
35         thrown.expect(ListAccessError.class);
36         thrown.expectMessage("Index out of bounds");
37         list.get(-2);
38     }
39

```

Week 8

```
public class BinaryTree<T extends Comparable<? super T>> implements BTree<T>
{
    TreeNode<T> root;
    @Override
    public void insert(T value)
    {
        //if the root is empty
        if(root==null)
        {
            //Add the new node to the root
            root = new TreeNode<T>(value);
        }
        //if the value compared to the root is less than zero (it's smaller
        than the root)
        else if(value.compareTo(value())<0)
        {
            //move to the left node, try insert again
            root.left().insert(value);
        }
        else
        {
            //move to the right node, try insert again
            root.right().insert(value);
        }
    }

    //Get root value
    @Override
    public T value()
    {
        return root.value;
    }
    //Get value of the left node
    @Override
    public BTree<T> left()
    {
        return root.left;
    }
    //Get value of the right node
    @Override
    public BTree<T> right()
    {
        return root.right;
    }
}
```



```

public static void main(String args[])
{
    {

        BinaryTree<Integer> tree = new BinaryTree<>();

        tree.insert(1);
        tree.insert(2);
        tree.insert(3);
        tree.insert(-1);
        Integer leftV = tree.left().value();
        Integer rightV = tree.right().value();
        Integer rightRightV = tree.right().right().value();
        System.out.println(tree.value());
        System.out.println(leftV);
        System.out.println(rightV);
        System.out.println(rightRightV);

    }
}

```

Results:

```

1
-1
2
3

```

Self-Assessment Week 1 – 8

Week	Overall	Docs	Struct	Names	Tests	Funct
1 & 2 Search timer	B	C	B	B	C	B
3 & 4 Generic swap	B	B	B	B	C	B
5 Sorting	B+	A	B	B	A	B
6 Linked lists	B+	B	B	A	B	B
8 Binary trees	B	B	B	B	C	B

Evidence/justification:

Week 1 & 2:

The CleverRandomListing performs the task required, the code is commented step by step so another user can see how it works which is shown in the documentation. The structure is easy to read with a simple to understand naming convention, I tested it's functionality with the timer multiple times and it works as desired. I feel the documentation can look busy and maybe could have done with some of the test results from the test class.

Week 3 & 4:

The Swap class swaps two objects in an array as required, it is generic as it swaps multiple types of objects, I demonstrated ints and strings swapping. I could have done a testing class rather than just a main but it does demonstrate it on a basic level. There are comments in the code to explain what's happening and I feel the naming was simple.

Week 5:

I have shown many results from the low sizes to high, each sheet having a description of what's happening. There are formulae to give an indication to how execution times vary to data sizes, however I feel the Quick Sorts algorithm is more original and accurate than the selection sorts.

Week 6:

The names of what the functions do are named clearly with comments, it performs the task which is demonstrated in the test methods, one shows a test with a large array the other shows a test with a small array. These have been shown with the results printed.

Week 8:

It performs the task as needed, could have made a detailed test class, however I have made a minitest that shows an example of it working. Could have done a larger test sample. Have included comment, naming is generic and easy to understand.

Week 9

When the values are added their position is determined by their ascii value of the key % of the size of the table, a formula for this could look like this:

$$x \% n$$

Where x is the ascii value of the key and the n is the length of the hash table

When adding values to the hash table once the size has reached the threshold the hash tables size increases, the hash tables size seems to increase by its own size by x2 to how many times it has increased. A formula of this is:

$$X * 2^n$$

Where X is the hash table size starting from 1 and n is the amount of times it has been increased.

Week 10

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

    @Override
    public List<T> traverse() throws GraphError
    {
        for(int i = 0; i<getNodes().size(); i++)
        {
            //If we have not visited all the nodes
            if(visited.size() < getNodes().size())
            {
                //store our current node
                @SuppressWarnings("unchecked")
                T startNode = (T) getNodes().toArray()[i];
                //If our visited array does not contain this node
                if(!visited.contains(startNode))
                {
                    recursiveDepthFirstTraversal(startNode);
                }
            }
            else break;
        }
        return ourGraph;
    }

    public void recursiveDepthFirstTraversal(T node) throws GraphError
    {
        //Add our node to the visited list and our graph
        visited.add(node);
        ourGraph.add(node);
        //set what the neighbours are
        Set<T> neighboursSet = getNeighbours(node);
        //put the neighbours into an object array
        @SuppressWarnings("unchecked")
        T[] neighbouringNodes = (T[]) neighboursSet.toArray();
        //check through all the neighbouring nodes
        for (int i = 0; i < neighbouringNodes.length; i++)
        {
            //this node is the current neighbour
            T n = neighbouringNodes[i];
            // if it is not visited and exists
            if (n != null && !visited.contains(n))
            {
                //start again from here
                recursiveDepthFirstTraversal(n);
            }
        }
    }
}
```

```

public static void main(String[] args) throws GraphError
{
    DepthFirstTraversal<Integer> graph = new DepthFirstTraversal<>();

    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);

    graph.add(node0);
    graph.add(node1);
    graph.add(node2);
    graph.add(node3);
    graph.add(node4);

    graph.add(0, 3);
    graph.add(0, 2);
    graph.add(1, 0);
    graph.add(2, 1);
    graph.add(3, 4);
    graph.add(4,0);

    graph.traverse();
    System.out.println("Recursive Depth First Traversal:
"+Arrays.toString((graph.ourGraph.toArray())));
}
}

```

Results:

Recursive Depth First Traversal: [0, 2, 1, 3, 4]

Week 11

```
public class ReferenceCountTopologicalSort<T> extends AdjacencyGraph<T> implements
TopologicalSort<T>
{
    //create our hashmap
    HashMap<T, Integer> map = new HashMap<T, Integer>();
    //create our list for sorted nodes
    List<T> sortedNodes = new ArrayList<T>();

    @Override
    public List<T> getSort() throws GraphError
    {
        initialise();
        setUpReferenceCounts();
        addToSort();
        System.out.println(map);
        System.out.println(sortedNodes);
        return sortedNodes;
    }

    private void initialise()
    {
        //add all our empty nodes
        for (T node : getNodes())
        {
            map.put(node, 0);
        }
    }

    private void setUpReferenceCounts() throws GraphError
    {
        //for every object in every node
        for (T node : getNodes())
        {
            //for every object in every neighbours node
            for (T successor : getNeighbours(node))
            {
                int references = map.get(successor);
                //if the current neighbour is not null
                if (map.get(successor) != null)
                {
                    //add to our hashmap
                    map.put(successor, ++references);
                }
            }
        }
    }
}
```

```

public void addToSort() throws GraphError
{
    //whilst we have not sorted the nodes
    while(sortedNodes.size() < getNodes().size())
    {
        for (T node : getNodes())
        {
            //if the node is not null and the value is more than 0
            if (map.get(node) != null && map.get(node).intValue() >= 0)
            {
                //add to our sorted nodes
                sortedNodes.add(node);
                //for every neighbour
                for (T successor : getNeighbours(node))
                {
                    Integer references = map.get(successor);
                    //if our successor is not null
                    if (references != null)
                    {
                        //add to our map
                        map.put(successor, references - 1);
                    }
                }
                //remove from our map
                map.remove(node);
                break;
            }
        }
    }
}

```



```

    public static void main(String[] args) throws GraphError
    {
        ReferenceCountTopologicalSort<Integer> graph = new
ReferenceCountTopologicalSort<>();

        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(0, 1);
        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);

        graph.getSort();
    }

```

Results:

```

{}
[3, 6, 0, 1, 2, 5, 8, 7, 9, 4]

```

Self-assessment Week 9 – 11

Week	Overall	Docs	Struct	Names	Tests	Func
9 Hashtable	C	Not applicable. Think about how well you have explained the hashtable's behaviour	Not applicable. Think about how well you have explained the hashtable's behaviour	Not applicable. Think about how well you have explained the hashtable's behaviour	Not applicable. Think about how well you have explained the hashtable's behaviour	Not applicable. Think about how well you have explained the hashtable's behaviour
10 Depth first traversal	B	B	B	B	C	B
11 Reference counting topological sort	B	B	B	B	C	B

Week 9:

I gave a concise description on how hashables work using formulae I came up with, I tried to explain this in an understandable way.

Week 10:

I feel it fulfils the function required however it lacks thorough evidence of testing, it is fully commented, it is structured clearly and uses a generic naming convention.

Week 11:

Similar to week 10, should have used a test class however does show its test and results. It uses generic naming convention, it is structured clearly and works as intended.

Week 13

- 1) The test is not always guaranteed to terminate. The test could theoretically always count between the two finishing points meaning it never reaches the end, though this is unlikely
- 2) The shortest amount of lines that can be achieved is 14, this is due to the counter starting at a finishing point, recognising this then counting up/down to the other finish point.
- 3) The largest number the counter can reach is 11, this is because it starts at 10 then immediately adds one.
- 4) The lowest possible number the counter can reach is -1 this happens when it starts as 0 and immediately counts down.

Week 14

```
public void runTrain() throws RailwaySystemError
{
    Clock clock = getRailwaySystem().getClock();
    Railway nextRailway = getRailwaySystem().getNextRailway(this);
    while (!clock.timeOut())
    {
        //start
        choochoo();
        //put a stone in my current track
        getBasket().putStone(this);
        // whilst the other track has a stone
        while (nextRailway.getBasket().hasStone(this))
        {
            //if we both have a stone
            if(nextRailway.getBasket().hasStone(this) ==
               getBasket().hasStone(this))
            {
                //pick up my stone so they can pass
                getBasket().takeStone(this);
                // They have not taken their stone so wait
                while(nextRailway.getBasket().hasStone(this)
                      != getBasket().hasStone(this))
                    siesta();
                //put the stone back, they're on the track
                getBasket().putStone(this);
            }
        }
        //change track
        crossPass();
    }
}
```

Test results:

```
Clock: tick tock
Peru: choo-choo
Bolivia: choo-choo
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: checking Peru's basket for stones
Peru: checking Peru's basket for stones
Bolivia: checking Bolivia's basket for stones
Clock: tick tock
Peru: removing stone from Peru's basket (1 stone in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
Peru: checking Peru's basket for stones
Bolivia: checking Peru's basket for stones
Peru: zzzzz
Peru: checking Bolivia's basket for stones
Bolivia: checking Bolivia's basket for stones
Peru: checking Peru's basket for stones
```

Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Clock: tick tock

Bolivia: checking Peru's basket for stones
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: entering pass
Bolivia: crossing pass
Bolivia: leaving pass
Bolivia: choo-choo
Peru: checking Bolivia's basket for stones
Bolivia: adding stone to Bolivia's basket (1 stone in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: checking Peru's basket for stones
Peru: checking Peru's basket for stones
Clock: tick tock

Bolivia: checking Peru's basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking Bolivia's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: removing stone from Bolivia's basket (2 stones in the basket)
Peru: checking Peru's basket for stones
Peru: zzzzz
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: checking Bolivia's basket for stones
Clock: tick tock

Peru: checking Peru's basket for stones
Bolivia: zzzzz
Peru: zzzzz
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: checking Bolivia's basket for stones
Peru: checking Peru's basket for stones
Bolivia: zzzzz
Peru: zzzzz
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Clock: tick tock

Peru: checking Peru's basket for stones
Bolivia: checking Bolivia's basket for stones
Peru: zzzzz
Bolivia: zzzzz
Peru: checking Bolivia's basket for stones
Bolivia: checking Peru's basket for stones
Bolivia: checking Bolivia's basket for stones
Peru: checking Peru's basket for stones
Bolivia: zzzzz
Peru: zzzzz
Clock: tick tock

Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: checking Bolivia's basket for stones
Peru: checking Peru's basket for stones
Peru: zzzzz
Bolivia: zzzzz
Peru: checking Bolivia's basket for stones
Bolivia: checking Peru's basket for stones
Clock: tick tock

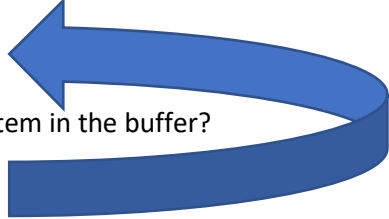
Peru: checking Peru's basket for stones
Bolivia: checking Bolivia's basket for stones

Peru: zzzzz
 Bolivia: zzzzz
 Peru: checking Bolivia's basket for stones
 Bolivia: checking Peru's basket for stones
 Peru: checking Peru's basket for stones
 Bolivia: checking Bolivia's basket for stones
 Bolivia: zzzzz
 Peru: zzzzz
 Peru: checking Bolivia's basket for stones
 Bolivia: checking Peru's basket for stones
 Clock: tick tock
 Peru: checking Peru's basket for stones
 Bolivia: checking Bolivia's basket for stones
 Peru: zzzzz
 Bolivia: zzzzz
 Peru: checking Bolivia's basket for stones
 Bolivia: checking Peru's basket for stones
 Peru: checking Peru's basket for stones
 Bolivia: checking Bolivia's basket for stones
 Clock: tick tock
 Peru: zzzzz
 Bolivia: zzzzz
 Bolivia: checking Peru's basket for stones
 Peru: checking Bolivia's basket for stones
 Peru: checking Peru's basket for stones
 Bolivia: checking Bolivia's basket for stones
 Peru: zzzzz
 Bolivia: zzzzz
 Peru: checking Bolivia's basket for stones
 Peru: checking Peru's basket for stones
 Clock: tick tock
 Bolivia: checking Peru's basket for stones
 Peru: zzzzz
 Bolivia: checking Bolivia's basket for stones
 Peru: checking Bolivia's basket for stones
 Bolivia: zzzzz
 Peru: checking Peru's basket for stones
 Bolivia: checking Peru's basket for stones
 Peru: zzzzz
 Clock: tick tock
 Bolivia: checking Bolivia's basket for stones
 Peru: checking Bolivia's basket for stones
 Bolivia: zzzzz
 Peru: checking Peru's basket for stones
 Bolivia: checking Peru's basket for stones
 Peru: zzzzz
 Bolivia: checking Bolivia's basket for stones
 Peru: checking Bolivia's basket for stones
 Peru: checking Peru's basket for stones
 Bolivia: zzzzz
 Clock: tick tock
 Peru: zzzzz
 Bolivia: checking Peru's basket for stones
 Peru: checking Bolivia's basket for stones
 Bolivia: checking Bolivia's basket for stones
 Bolivia: zzzzz
 Peru: checking Peru's basket for stones
 Bolivia: checking Peru's basket for stones ----- This is ongoing

Week 15

1)

```
public T get() throws BufferError, SemaphoreLimitError {  
    T item;  
    try {  
        criticalSection.poll(); // is the buffer available?  
        noOfElements.poll(); // is there at least one data item in the buffer?  
        // criticalSection.poll(); // is the buffer available?  
        item = getItem(); // add the data item  
        criticalSection.vote(); // make the buffer available again  
        noOfSpaces.vote(); // there is now one more space in the buffer  
    } catch (InterruptedException ie) {  
        throw new BufferError("Buffer: Data item could not be retrieved from the buffer.\n" +  
            "\t" + ie.getMessage());  
    }  
    return item;  
}
```



Swapping the order of the `criticalSection.poll()` and the `noOfElements.poll()` causes an error situation

2) The error occurs as there are no permits available, so the Semaphore waits until one is available, which never occurs so it just stops after the 20 seconds as it is set to do. The original code starts with permits available, so it can proceed.

Put	Get	noOfSpaces	noOfElements	criticalSection
		10	0	1
	Critsec.poll()	10	0	0
noOfSpaces()		9	0	0
Critsec.poll()		9	0	0
END				

Put	Get	noOfSpaces	noOfElements	criticalSection
		10	0	1
	noOfElements.poll()	10	0	1
noOfSpaces()		9	0	1
Critsec.poll()		9	0	0
Critsec.vote()		9	0	1
noOfElements()		9	1	1
	Critsec.poll()	9	0	0
	Critsec.vote()	9	0	1
	noOfSpaces.vote()	10	0	1

3) It is not essential in the put() as shown in the table below.

Put	Get	noOfSpaces	noOfElements	criticalSection
		10	0	1
	noOfElements.poll()	10	0	1
Critsec.poll()		10	0	0
noOfSpaces()		9	0	0
Critsec.vote()		9	0	1
noOfElements()		9	1	1
	Critsec.poll()	9	0	0
	Critsec.vote()	9	0	1
	noOfSpaces.vote()	10	0	1

Week 16

```
public class LockResourceManager extends BasicResourceManager
{
    //new lock
    final Lock lock = new ReentrantLock();

    //create our array of conditions for each priority
    final Condition[] conditions = new Condition[11]; //lock.newCondition();

    boolean inUse = false;

    public LockResourceManager(Resource resource, int maxUses)
    {
        super(resource, maxUses);
        //initialise our conditions array
        for(int i = 0; i < conditions.length; i++)
        {
            conditions[i] = lock.newCondition();
        }
    }

    public void requestResource(int priority) throws ResourceError
    {
        lock.lock();
        try
        {
            //if our resource is inuse
            if(inUse)
            {
                //add to the number waiting
                increaseNumberWaiting(priority);
                //wait until resource is free
                conditions[priority].await();
            }
            // resource is in use
            inUse = true;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        finally
        {
            lock.unlock();
        }
    }
}
```

```

public int releaseResource() throws ResourceError
{
    lock.lock();
    try
    {
        for(int i = 10; i >= 0; i--)
        {
            //if anything is waiting
            if(getNumberWaiting(i) > 0)
            {
                //decrease the number waiting
                decreaseNumberWaiting(i);
                //signal to tell user it can stop waiting
                conditions[i].signal();
                //resource is no longer in use
                inUse = false;
                //return our user
                return i;
            }
        }
        //not in use
        inUse = false;
        //no one is waiting
        return NONE_WAITING;
    }
    finally
    {
        lock.unlock();
    }
}

public void test4_1() throws ResourceError
{
    ResourceSystem resource1 = new ResourceSystem();
    // The resource - may be used up to 20 times
    resource1.addResource("A", 20);
    // User 1 uses the resource for up to 1/10 second each time
    resource1.addUser("1",0.1);
    // User 2 uses the resource for up to 1/10 second each time
    resource1.addUser("2",0.1);
    // User 3 uses the resource for up to 1/5 second each time
    resource1.addUser("3",0.2);
    // User 4 uses the resource for up to 1/5 second each time
    resource1.addUser("4",0.2);
    resource1.run();
}

```

Results:

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

Self-Assessment Week 13 – 16

Week	Overall	Docs	Structure	Names	Tests	Func
13 Counter Behaviour	C	N/A	N/A	N/A	N/A	N/A
14 Dekker Trains	A	B	B	A	A	A
15 Semaphore Behaviour	A	N/A	N/A	N/A	N/A	N/A
16 Locks and Conditions	B	B	B	B	B	C

Week 13:

I did identify important aspects such as the largest and shortest number and shortest length however these did not go into enough detail.

Week 14:

I feel it does what it needs to do effectively, it works every time and I have added the results to the logbook, the names follow the convention set even in the comments which describes step by step, I feel there isn't much more I could have done.

Week 15:

I feel I went into a good amount of detail with tables to show it's progression clearly step by step. I think I have demonstrated my strong understanding of semaphores in this work.

Week 16:

I believe it works as intended, the tests are shown clearly, the names are generic, the structure I feel is tidy and it is fully commented.

Week 17

```
AND = [  
    1 1 1 0  
    0 0 0 1  
];
```

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

```
OR = [  
    1 0 0 0  
    0 1 1 1  
];
```

```
%kron(NOT, AND) = A (Tensor Product of Not & And)
```

```
A = [  
    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];
```

```
%OR * A = B
```

```
B = [  
    0    0    0    0    1    1    1    0  
    1    1    1    1    0    0    0    1];
```

Week 20

```
%A is ONE or ZERO

ZERO = [1;0];

ONE = [0;1];

Hadamard = (1/sqrt(2))*[1,1;1,-1];

%Hadamard * ONE = B1

B1 =[0.7071
     -0.7071];

%Hadamard * ZERO = B0

B0 =[0.7071
     0.7071];

%State C is either of the below
%Hadamard * B1  = ONE

%Hadamard * B0 = ZERO

%If either are these are run through a Hadamard gate twice they return to
%their original value

%If someone was using probabilities it would be less accurate although
%easier, it would start 100% ONE or ZERO, then it'd be 50% ONE or ZERO then
%be 100% ONE or ZERO again. whereas the matrix, you can follow whether
%it'll be either one as the B1 has a -0.7071 where as B0 does not.

%The probabilities make reversal impossible due to the loss of information
```

Self-Assessment Week 17 – 20

Week	Topic	Grade	Criteria
17	Modelling Circuits	C	Have you derived a matrix for a half-adder? Have you correctly applied the matrix methods for constructing sequential and parallel circuits? Have you explained/justified/proved your derivation? Have you tested it on (matrix representations of) various inputs?
20	Quantum Computing	A	Have you fully analysed the values that will appear at points A, B, and C in the circuit? Have you discussed the relationship between the values appearing at A and C? Have you considered what the implications of a purely probabilistic model would be for maintaining this relationship?

Week 17:

I feel I have completed the task using MATLAB, showing my workings out, however I could have gone into more detail of how it worked and done more testing.

Week 20:

I have shown the values at point A, B and C, I have discussed the relationship between the values A and C which is they are the same and I have shown why this is more accurate than a probability model due to loss of data.