Algorithms Processes and Data Logbook

Wasim Ramzan (u1970064)

January 23, 2021

Contents

1   Unit 1: Welcome

This is my logbook for the Algorithms Processes and Data Module. Here, you can find details about the work I have done throughout the modules.
No exercises set for this unit.

2   Unit 2: Tools

No exercises set for this unit.

3   Unit 3: Recap / CleverSearcher

I have implemented the CleverSearcher by using a small array (helper array) to compare with the original array. Depending on the results, elements will be swapped if the original array is larger than the small array. Then the small array will be sorted if changes have been made.

3.0.1   Timings

I tested the time it took both tests to reach a size of 100,000,000. My findings were that the CleverSearcher managed to reach a size of 100,000,000 substantially quicker than the SimpleSearcher. Furthermore, the Simple-Searcher ended at a size of 70,000,000 as the 5 second time limit triggered.

The graph below shows the SimpleSearcher slowing down around a size of 1,000,000, then ends at 70,000,000 as previously mentioned. SimpleSearcher reaches just below the 6 second mark. On the other hand, CleverSearcher stays well below the 1 second mark and managed to reach a greater size compared to the SimpleSearcher. This proves CleverSearcher being both efficient and faster.
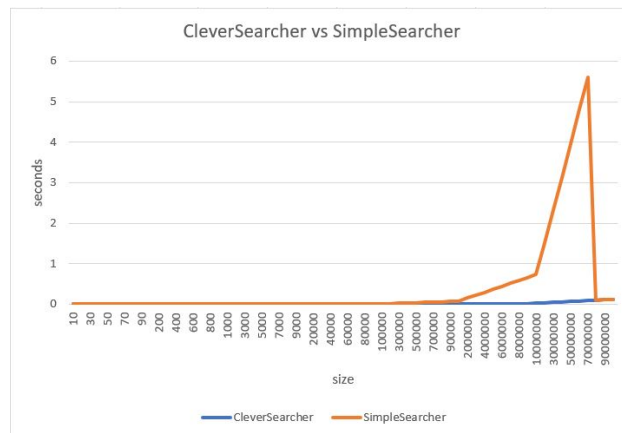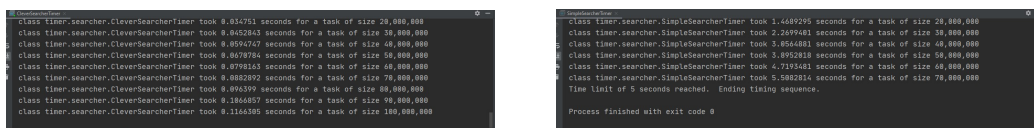

Figure 1: Graph Comparison

Figure 2: CleverSearcher and SimpleSearcher Timer Outputs

3.0.2 CleverSearcherTest

The CleverSearcherTest can be used to check if the index and array size are within range. If they are it will return true. These are tests to check how the program reacts when the index is both out of range as well as in range.

- The tests minus1in10, 10001in10000, 0in0 fail because they are not within the bounds. For example, test 0in0 would not be able to return the index as there is nothing in the array. Likewise, tests minus1in10 and 10001in10000 cannot find the index as they are not within the array size.

- Tests 3in10 and test999999in1000000 pass because they are within the range of the array as well as the index ranges.

- LargeNumber stops as the "array size exceeds VM limit". It could not handle the large numbers so simply stopped rather than returning fail or true. It would however return true if able to run - as it is within the bounds of the index

** Code can be viewed on Unit 3 files **

3.0.3 CleverSearcherTest Output
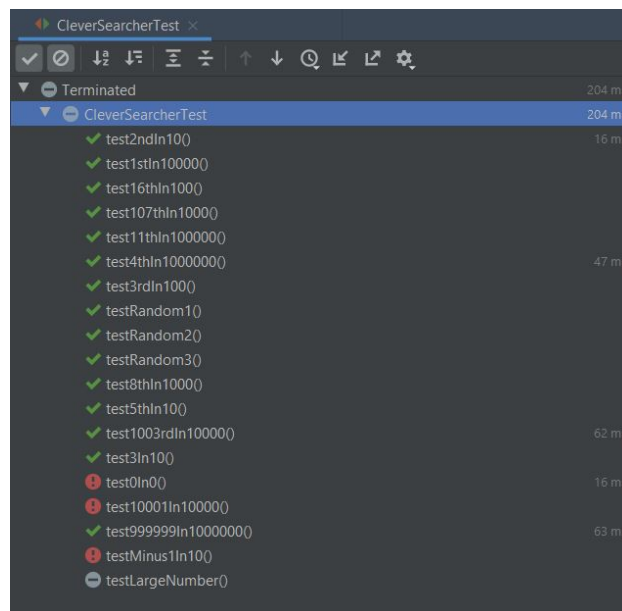
Figure 3: CleverSearcherTest Output

Note: As mentioned, some tests failed as they are out of range, which is to be expected.

4 Unit 4: Testing

No exercises set for this unit.

4.1 Self-Assessment (Units 3 & 4)

Grade: 5/5
Comments: The reason I gave myself 5/5 is because I documented my findings in depth. Furthermore, I have completed all the tasks provided as well as implementing further tests of my own.
Using my tests, I gave examples of what the results would be and why. As for the methods, I provided analysis of both CleverSearcher and SimpleSearcher in terms of their speed and efficiency.

5 Unit 5: Correctness / Cube Program

5.0.1 Cube Code with Assertions

```java
public static int cube(int n) {
    int cube = 0, threeNsquared = 0, threeN = 0;
    int i = 0;
    assert cube == i*i*i : cube;
1   assert threeN == 3*i : threeN;
    assert threeNsquared == 3*i*i;
    while (i < n) {
        assert cube == i*i*i : cube;
2       assert threeN == 3*i : threeN;
        assert threeNsquared == 3*i*i;
        cube = cube + threeNsquared + threeN + 1;
        assert cube == (i+1)*(i+1)*(i+1);
3       assert threeN == 3*i;
        assert threeNsquared == 3*i*i;
        threeNsquared = threeNsquared + 2*threeN + 3;
        assert cube == (i+1)*(i+1)*(i+1);
4       assert threeN == 3*i;
        assert threeNsquared == 3*(i+1)*(i+1);
        threeN = threeN + 3;
        assert cube == (i+1)*(i+1)*(i+1);
5       assert threeN == 3*(i+1);
        assert threeNsquared == 3*(i+1)*(i+1);
        i++;
        assert cube == i*i*i;
6       assert threeN == 3*i;
        assert threeNsquared == 3*i*i;
    }
    assert cube == n*n*n : cube;
7   assert threeN == 3*n;
    assert threeNsquared == 3*n*n;
    return cube;
}
```
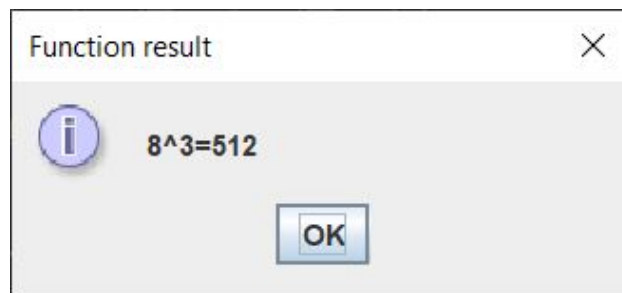

Figure 4: Cube output test

5.0.2 Cube Elucidations

Assertion No.	Elucidation
1	cube is i^3, threeN is 3 * i, threeNsquared is 3 * i * i. Maintaining this assertion throughout to prove the method returns n^3.
2	this assertion is identical to assertion 1 therefore it must be true.
3	cube is now (i+1) * (i+1) * (i+1). This is to prepare for the incrementation on line 23.
4	threeNsquared is now 3 * (i+1) * (i+1). This is also to prepare for the increment on line 23.
5	threeN is now 3 * (i+1). This is to prepare for the increment on line 23.
6	see 1
7	the assertions from 1 hold. However, i has been replaced with n, and cube = n^3. This proves the method is correct as cube is set as the return value.

5.0.3 Cube Justifications

Assertion No.	Justification
1	The assignments on line 2 & 3 state that cube = 0, threeN = 0, threeNsquared = 0 and i = 0. So, the assertion states three times that $0 = 0$ which is true.
2	The only routes to this assertion are through 1 and 6. These assertions are identical so if they are true then assertion 2 will also be true.
3	This assertion depends on the assignment on line 11, impacting the value of cube. To further justify this, cube will represent the value of cube prior to the assignment, whereas cube' will represent the value after the assignment. cube = i^3, threeN = 3i, threeNsquared = $3i^2$ **(assertion 2)** cube' = cube + threeNsquared + threeN + 1 cube' = $(i+1)^3$, threeNsquared = $3i^2$, threeN = 3i **(assertion 3)** Justification: The assertion on line 11 does not change the values of threeNsquared and threeN. So, the path will be: cube' = cube + threeNsquared + threeN + 1 **(from assignment on line 11)** = $i^3 + 3i^2 + 3i + 1$ **(from assertion 2)** = $(i + 1)^3$ **(as $(i + 1)^3 = i^3 + 3i^2 + 3i + 1$)**
4	The assignment on line 15 does not change the value of cube and threeN from assertion 1. The first part follows from assertion 4, and I will use the same justification concept as cube and cube'. threeNsquared' = threeNsquared + 2 * threeN + 3 **(assignment on line 15)** = $3i^2 + 2 * 3i + 3$ **(from assertion 3)** = $3 * (i + 1)^2$
5	The assignment on line 19 does not change the value of cube and threeNsquared from assertion 1. I will use the same justification concept using threeN and threeN' following from assertion 5. threeN' = threeN + 3 **(from assignment on line 19)** = $3i + 3$ **(from assertion 4)** = $3 * (i + 1)$
6	Using i' for the value of I after the increment on line 23: cube = $(i + 1)^3$, threeN = $3(i + 1)$, threeNsquared = $3(i + 1)^2$ **(from assertion 5)** = i'^3 = 3i' = $3i'^2$ i' = (i + 1) because of line 23's increment
7	To reach this point it means that the test has found that i is equal to n. This means the test has passed and the only way to this route is through assertions 1 to 6. If all these assertions are true then it is confirmed that cube = i^3, threeN = 3i and threeNsquared = $3i^2$. So, the result returned by the algorithm will be n^3.

5.1 Self-Assessment

Grade: 5/5
Comments: For this unit I provided all the necessary information regarding the cube program. I gave the code with assertions, elucidations as well as justifications. The main reason I deserve this mark is because of the amount of depth I went into both the elucidations and justifications. Within the justification, I gave clear reference to the internal findings of the cube program when providing my justifications.

6 Unit 6: Generics

6.1 Description

The following methods have been implemented:
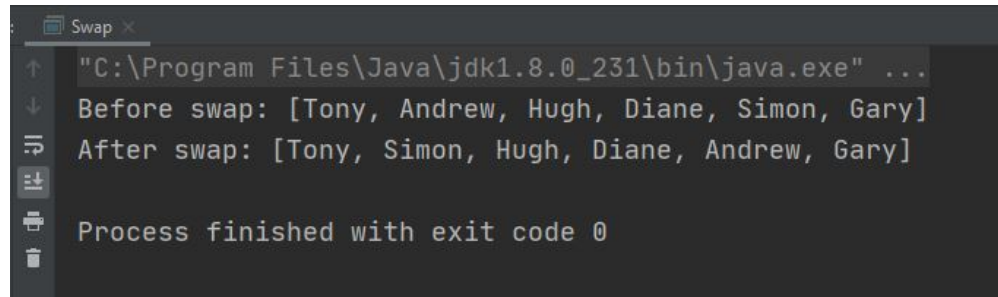
- Swap and Max (genericMethods) with tests

- IsEven, IsOdd, IsPalindrome and IsPrime with tests

- CountingUnaryPredicate with classes above inheriting the numberSatisfying method.

- IsMonotonic not yet implemented

** These can be found on the Unit 6 Files **

Out of these, IsPalindrome is the one suitable to work with both integers and strings.

6.2 Swap Result

Swap

Figure 5: Swap Method

6.3 IsEven, IsOdd, IsPalindrome and IsPrime Tests

The full test classes can be found on my Unit 6 files.
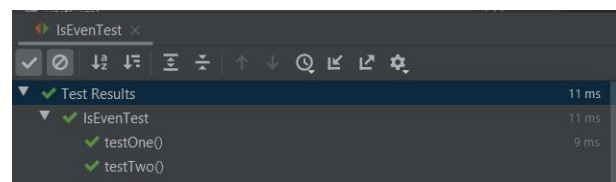
IsEven Example Test

For this there are series of tests to check whether the given number is even. For example, 'testZero' will pass as the number is even, whereas 'testOne' will also pass as it returns false for the number 1. Further numeric tests can be found on the program.

Figure 6: IsEven example Test and Result

6.4 Self-Assessment

Grade: 5/5
Comments: I completed the Swap method which is working and implemented further methods as mentioned. Also, I went further and implemented test classes for these unaryPredicate classes. Although, I missed the extra isMonotonic method, it was not reasonable for me to deduct a point as I completed the other ones with no issues.

7 Unit 7: Searching

** Boyer Moore Implemented **

8 Unit 8: Sorting

8.1 Sorting Algorithms

Bubble Sort
As you can see on Figures 7 and 8 below, the bubble sort increases in time considerably at an array size of 10,000. Hence, there is definitely a relation between the array size and time. Furthermore, as the array size increases so will the time.

Bubble Sort formula
To create my approximate formula, I grouped the array size as arriving at an approximate formula for the whole data set proves to be difficult. This is due to how much the time varies and changes as the size increases. Note that this is highly inaccurate but works closely in conjunction with my setup. I worked with the IntegerBubbleSort results to form the formulas.

Bubble Sort formula (Size 1 to 1000)

$$\sum_{i=1}^{9} [i * 10^3)] \tag{1}$$

Bubble Sort formula (Size 2000 to 10,000)

$$\sum_{i=2000}^{10,000} [i * 15^3] \tag{2}$$
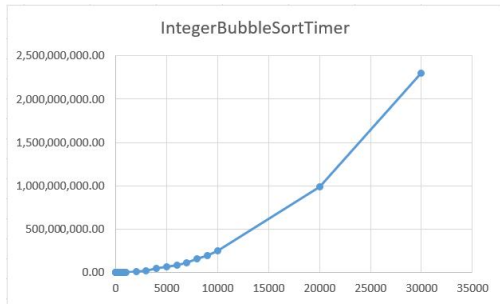
Figure 7: IntegerBubbleSortTimer
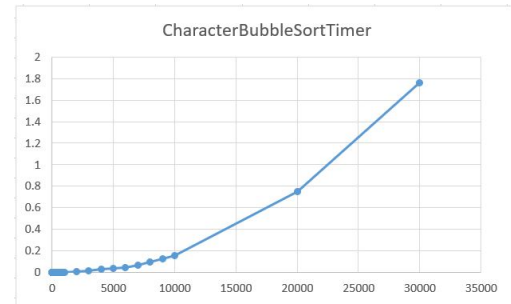
Figure 8: CharacterBubbleSortTimer

8.1.1 Bubble Sort Big O Analysis

The implemented bubble sort algorithm has a time complexity of $O(n^2)$. For N number of iterations we have to both compare and swap elements if required. The number of iterations results in:

$$(N-1) + (N-2) + (N-3) + ... + 3 + 2 + 1 = \frac{N(N-1)}{2} = \mathcal{O}(n^2) \text{ (Datta, 2020)}$$

Hence, the bubble sort algorithm is a quadratic algorithm.

8.2   Other Algorithms

I have implemented further algorithms such as:

- Selection sort $O(n^2)$

- Quicksort $O(n \log n)$

- Shell sort

** These can be found on the Unit 8 Code **

8.2.1   Other Algorithms Comparison
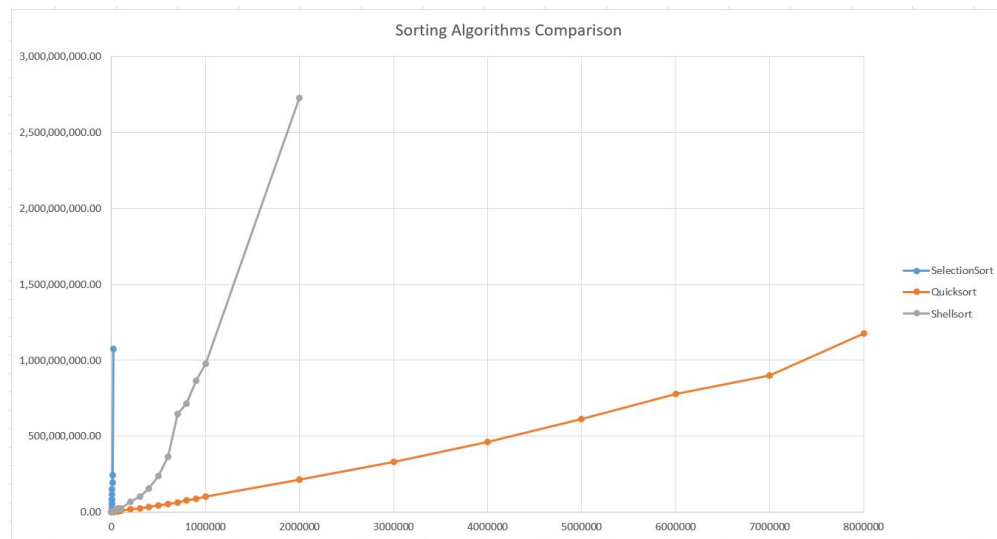
Figure 9: Other Algorithms

Other Algorithms Results Explanation

In Figure 9, quicksort is clearly the winner here as it reaches a larger array size whilst maintaining a decent execution speed. On the other hand, selectionsort is the slower out of these and is unable to reach greater than an array size of 20,000.
The algorithm I chose to implement (shellsort) is better than selectionsort as it reaches a greater array size and starts to become faster than selectionsort as the array size increases.

8.3 Bubble sort vs Insertion Sort

The comparisons on Figures 10 and 11 show that insertion sort is faster than bubble sort. Firstly, bubble sort time is higher than insertion sort, hence it is much slower. Also, insertion sort reaches the time limit at 30,000 whereas insertion sort reaches 40,000. Despite insertion sort reaching a higher array size it still manages to have a lower time than bubble sort.

The difference begins to show more as the size increases as to which one is the faster, although there may not be a massive difference.

Insertion Sort Formula

Bubble Sort formula was done before. So, in comparison to the previous formula they are rather similar although Insertion Sort is faster in the early sizes. When the size increases the difference between the two are approximately 3 times with insertion sort being faster. So, the formula between size 2000 and 10,000 was divided by 3. Like the previous formula this was done using IntegerInsertionSort:

Insertion Sort Formula (Size 1 to 1000)

$$\sum_{i=1}^{9}[i * 10^3)] \tag{3}$$

Insertion Sort formula (Size 2000 to 10,000)

$$\sum_{i=2000}^{10,000}[i * (15^3/3)] \tag{4}$$
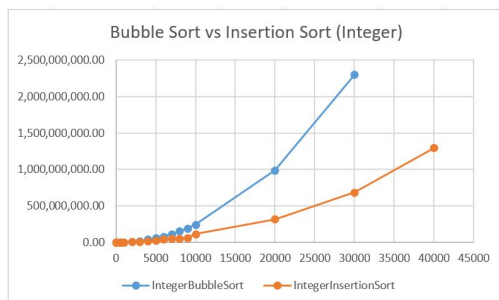
Figure 10: Bubble Sort vs Insertion Sort 1
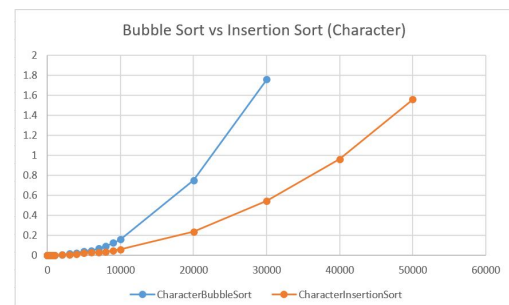
Figure 11: Bubble Sort vs Insertion Sort 2

8.4 Self-Assessment

Grade; 4/5

Comments: This unit I went quite in depth regarding the sorting algorithms and providing comparisons between them. The graphs give an excellent visual view of which algorithms are fast and slow. The reason I deducted a point is because I only mentioned big O notation for the bubble sort algorithm. Instead, I should have mentioned big O notation for all the algorithms I implemented.

9 Unit 9: Complexity

9.1 Complexity Analysis

The graph below shows my results after timing the various complexity methods. Firstly, I used a base size of 10 for all the tests as well as using various delays where necessary. For example, I used a delay of 6ms for the Logarithmic method etc. I capped my results at a size of 100 as I believe it was enough to show which method was most efficient.

As you can see on the figure 10's graph, the rank from most to least fast is:

1. Logarithmic $O(\log n)$

2. Linear $O(n)$

3. NlogN $O(NlogN)$

4. Quadratic $O(N^2)$

5. Cubic $O(N^3)$

6. Exponential $O(2^N)$

The methods Exponential, Cubic and Quadratic were not able to reach the size of the other methods as either they weren't capable of doing or a time limit was reached. The other three methods NlogN, Linear and Logarithmic are capable of going even higher than the size of 100 I set, and the difference between the three will gradually continue to increase. The graph shows Linear and Logarithmic being close, however as the size increases these two will become more apart, with logarithmic being superior.

* Unit 9 Methods Implemented *
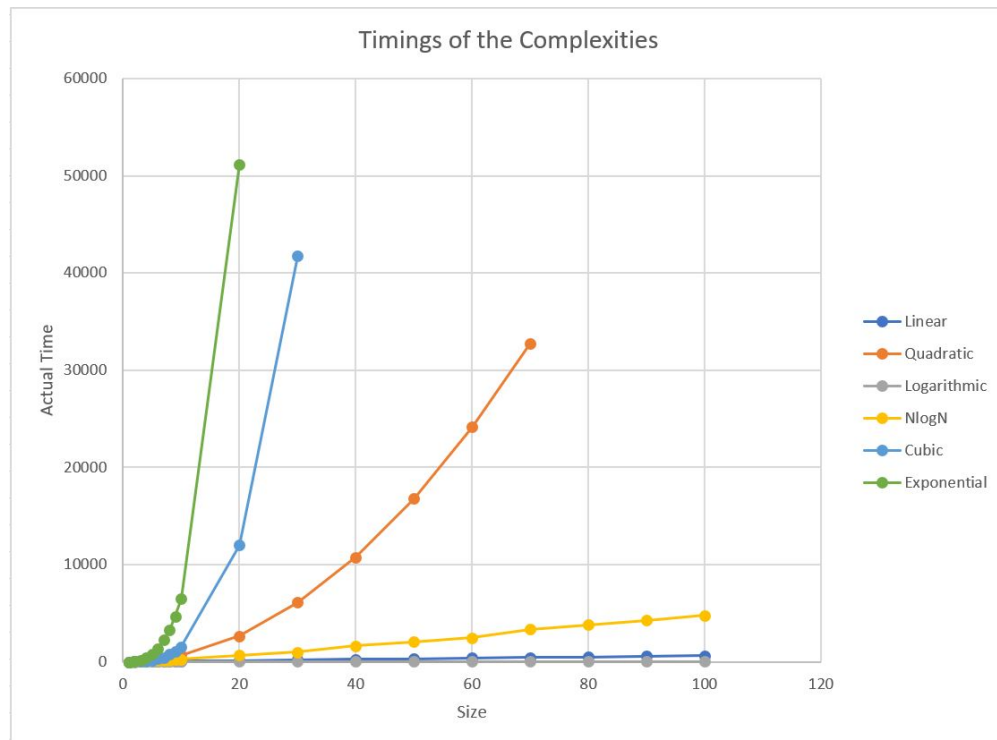
Figure 12: Timings of the Complexity Methods

9.2 Self-Assessment

Grade: 5/5
Comments: I have correctly implemented both Logarithmic and Cubic methods, along with their complexity. In addition, I have also implemented further complexities such as NlogN, Quadratic and Exponential. All of these have been plotted on a graph to provide a comparison and an explanation which I have provided.

10 Unit 10: Pointers

No Logbook Exercises.

11 Unit 11: Linked List

** Unit 11 SingleLinkList Methods Implemented **
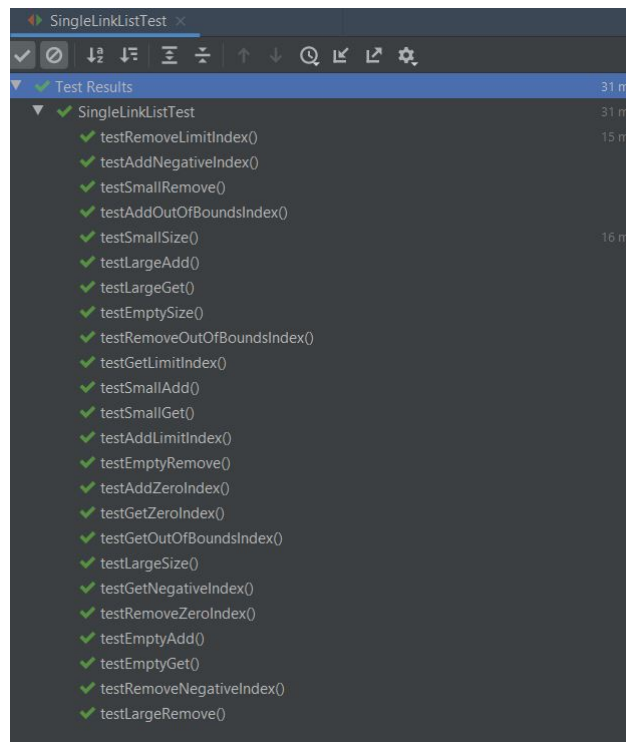
Figure 13: Graph Comparison

11.1 Self-Assessment

Grade: 4/5
Comments: I have only implemented the singlelistlist methods for this unit. Also, throughout creating my methods I ensured that boundary checking was used, to ensure correct values will be used. A point was deducted as I didn't create the test suite (one was provided), so I used that as I believed it to already be extensive.

12 Unit 12: Binary Trees

** Unit 12 Binary Tree Methods Implemented **
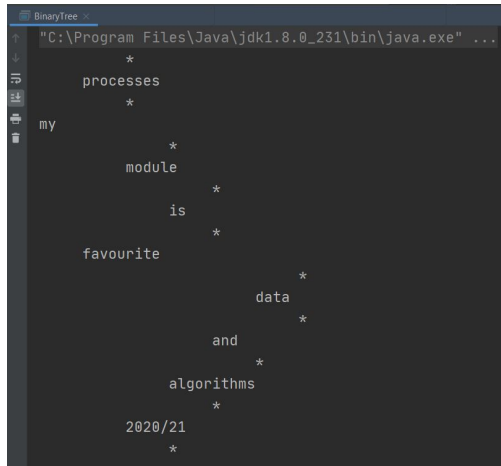
Figure 14: BTree Output
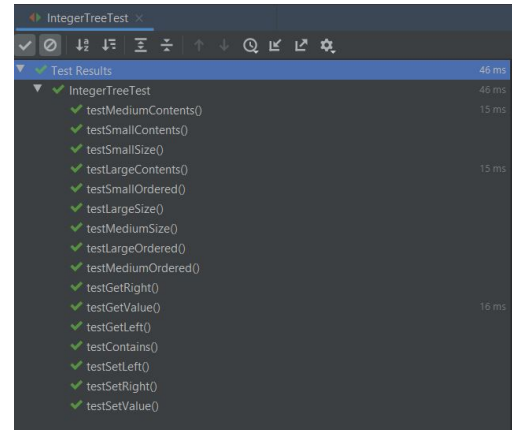
Figure 15: BTree Test

12.1 Self-Assessment

Grade: 3/5

Comments: I simply implemented the binary tree program and didn't create a test (one was provided) and haven't provided any documentation. To further improve, I could have done these or made an AVL tree/safe version of the setter methods. Hence, I deducted 2 points to compensate for this.

13 Unit 13: Hashtables

13.1 Hashtable Wrapper

To explain the behaviour of the hash table when the values have been added, I will explain what occurs when each value pair has been added and what changes were made to the hash table.

<u>Before</u>
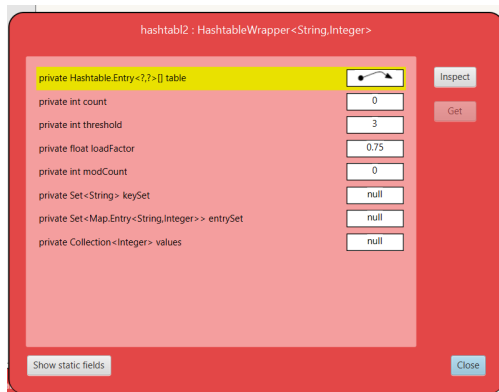
Figure 16: Hash Table Before 1
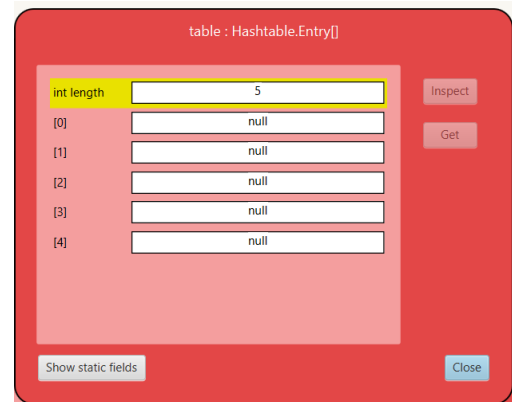
Figure 17: Hash Table Before 2

<u>("fred",37)</u>

Change #1 - private int count and private int modCount increased by 1.
Reason - As an entry has been added the count increases by 1.

Change #2 - value has been added to index of [2].
Reason - Using ASCII, the first letter "f" is 102. Hence, 102 % 5 = 2. So it is added to index [2].

Change #3 - Assigned hash of 3151467, object key "fred" and object value "37".
Reason - The hash is generated by using the hash(s) function. In this case the calculation will be:
ascii('f') $\times 31^3$ + ascii('r') $\times 31^2$ + ascii('e') $\times$ 31 + ascii('d') = 3151467.

<u>("is",69)</u>

Change #1 - private int count and private int modCount is now 2 after increasing by 1 again.
Reason - As an entry has been added the count increases by 1 resulting in a total of 2.

Change #2 - value has been added to index of [0].
Reason - Using ASCII, the first letter "i" is 105. Hence, 105 % 5 = 0. So it is assigned to index [0].

Change #3 - Assigned hash of 3370, object key "is" and object value "69".
Reason - The hash is generated by using the hash(s) function. In this case the calculation will be:
ascii('i') $\times$ 31 + ascii('s') = 3370.

("dead",0)

Change #1 - private int count and private int modCount is now 3 after increasing by 1 again.
Reason - As an entry has been added the count increases by 1 resulting in a total of 3.

Change #2 - value has been added to index of 3.
Reason - Using ASCII, the first letter "d" is 100. Hence, 105 % 5 = 0. [0] has an entry therefore we use quadratic probing, $(0 + 2^3)$ % 5 = 3.

Change #3 - Assigned hash of 3079268, object key "dead" and object value "0".
Reason - The hash is generated by using the hash(s) function. In this case the calculation will be:
ascii('d') x 31^3 + ascii('e') x 31^2 + ascii('a') x 31 + ascii('d') = 3079268.

("but",999)

Here is where things start to change. The private int count is now 4 however the modCount is 5. Also, the private int threshold increases from 3 to 8. The reason for the increase in threshold is to accommodate more entries as we surpassed the limit of 3, therefore it almost doubled the threshold. When we inspect the array, it shows the int length has increased from 5 to 11. On the other hand, the reason for the modCount to have increased by 2 is one for the entry, and another gets added whenever the threshold increases due to loadfactor and poor naming conventions within the program. As well as this, the assignments of entries have also changed:

1. ("fred",37) is now assigned to [0]

2. ("is",69) is now assigned to [4]

3. ("dead",0) is now assigned to [5]

4. ("but",999) is now assigned to [10]

This occurs due to the increase in large capacity resulting in the contents being rehashed and loaded into the new array. It looks like it has been resorted automatically by order of insertion.
The newly added ("but",999) value once inspected still has the object key for "but" and object value of 999 with a hash value of 97921. It is however assigned to [10].

"not",-42

This has a private int count of 5 and modCount of 6. Inspecting the array reveals it has been assigned to [4], object key of "not", and value -42 with a hash of 109267. There is also a change where the 'Hashtable.entry next' has a value. When I inspect this, it shows the "is,69" entry. Further analysis shows that "is,69" no longer has a separate entry in the table. So, in a sense both ("not",42) and ("is",69) have combined to use the [4] entry. The reason for this is to avoid collisions when both values are mapped to the same index.

("me!",-1)

The private int count is now 6, modCount of 7. It has been assigned to [3], 107913 hash and object key is "me!", with object value of -1. This seems to have mapped to a random index.

After
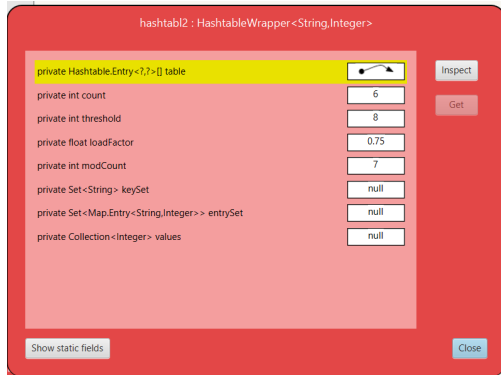
Figure 18: Hash Table After 1
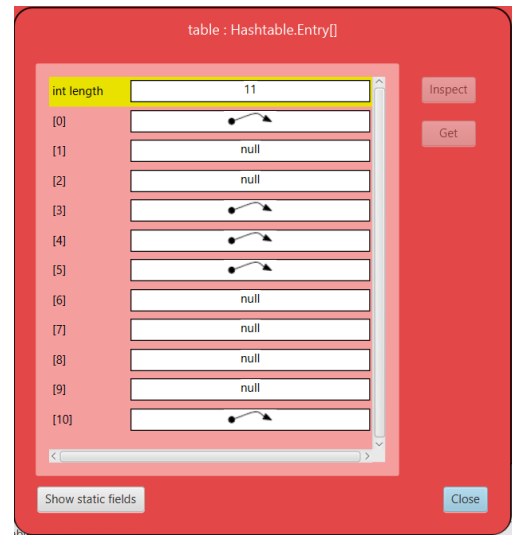
Figure 19: Hash Table After 2

13.2   Best Hash Function and Collisions

Question: Which of the following hash functions best avoids address collisions in a hash table with 100 elements (ensure you give reasons for your answer):

```java
public int hash1(String key) {
    return (int)(Math.round(Math.random()  100));
}

public int hash2(String key) {
    return key.length() % 100;
}

public int hash3(String key) {
    int midpnt = key.length() / 2;
    return (key.charAt(0)
        + key.charAt(midpnt)
        + key.charAt(key.length()-1));
}

public int hash4(String key) {
    int midpnt = key.length() / 2;
    return ((key.charAt(0)
        + key.charAt(midpnt)
        + key.charAt(key.length()-1)) % 100);
}
```

Out of the provided hash functions, **hash3** will be best to prevent collisions from occurring. This is because it is most likely to be unique after converting the first, mid, last character into ascii then adding them up.

Why not the other hash functions?

1. **hash1** has a high risk of collisions as there is a chance of having the same random number between 1-100.

2. **hash2** is in my opinion the worst out of these functions as if you have the same length string there will be numerous collisions.

3. **hash4** this is similar to hash3, however you find the remainder after dividing by 100. The reason why this one is worse is because if you add up the digits for two strings and for example have two results 128 and 228, the remainder will be 28 for both. Furthermore, it will be better to leave them in their original state.

13.3 Open Quadratic Hashtable

** OpenQuadraticHashtable implemented **

13.3.1 OpenQuadraticHashtable Test Result

By using the OpenHastableTest provided I extended this to test my QuadraticHashtableTest and have passed all eleven tests.
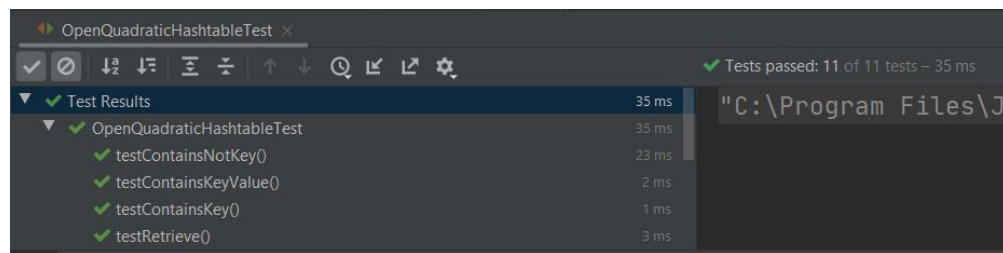
Figure 20: OpenQuadraticHastable Results

13.4 Self-Assessment

Grade: 5/5
Comments: I have gone in depth with explaining how the hashtables work and examining the internal array. Providing clear examples and describing the process throughout, I have done well to explain this. Also, in part 13.2 I answered the extra question where I describe the hash functions which was the best and why. Also, OpenQuadraticTable was implemented along with a test.

14 Unit 14: Graphs

** Depth-First Traversal Implemented **

14.1 Depth-First Traversal Test
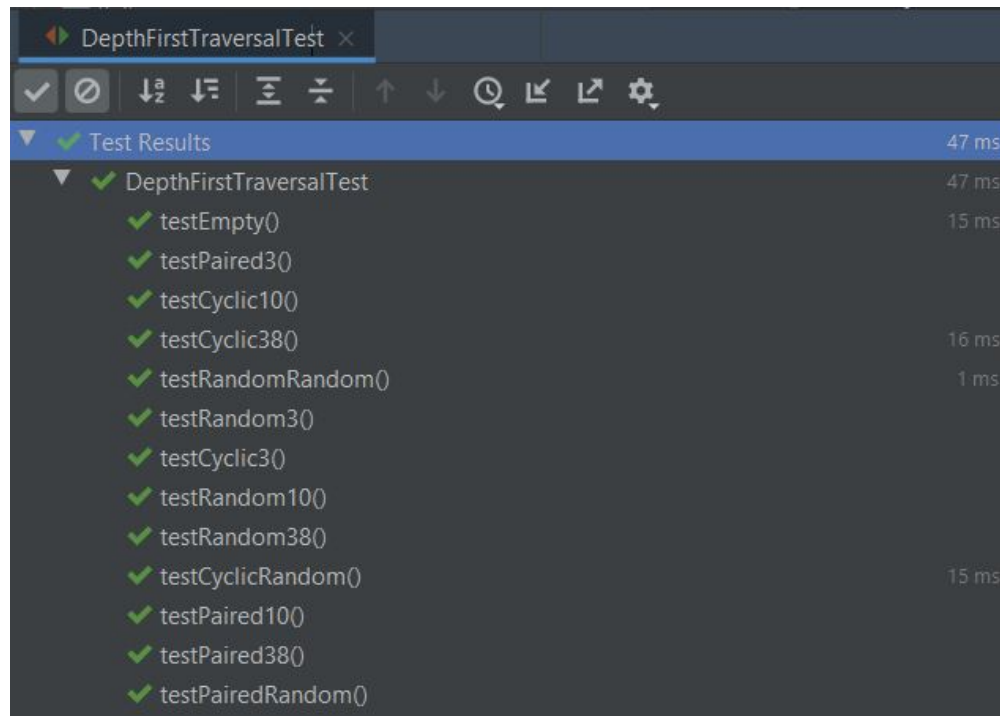
Figure 21: DepthFirstTraversal Test

14.2 Traversal Test Methods

- Cyclic Graph: These are tests for a graph that has one or more edges where the first and last vertices are equal. Testing these with random sizes ensures the traversals work with various ranges.

- Paired Graph

- ErdosRenyi : This is a model created and, in this case, defines a random generated graph. So, in the tests it generates a random amount of edges and will test these using the traversals.

14.3 Self-Assessment
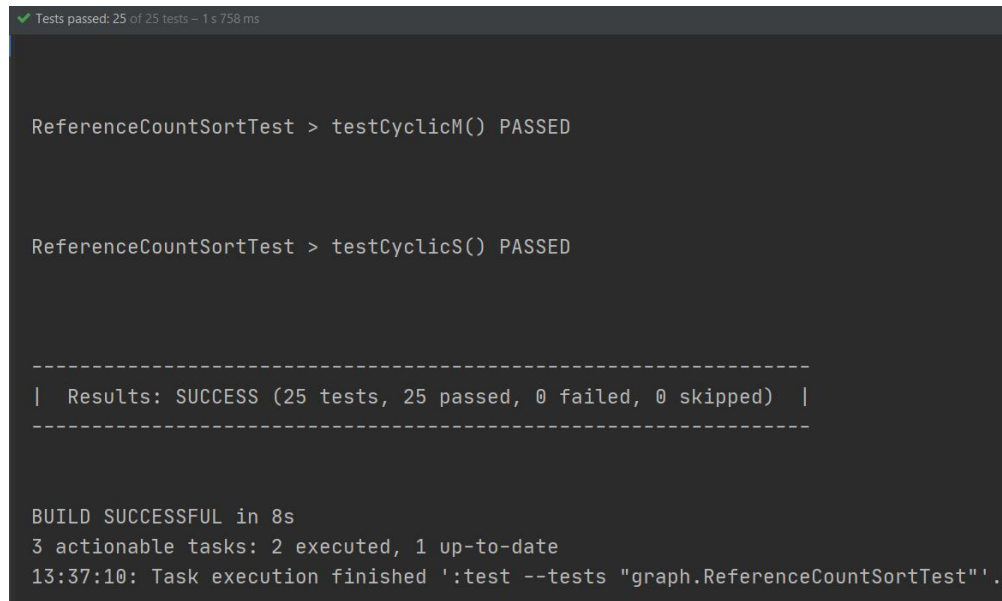
Grade: 4/5
Comments: Depth-first traversal has been implemented and is Javadoc compliant. Although there is a test for this class, it is using the TraversalTest which was provided (no improvements to the tests). To compensate for this, I deducted a point.
I have briefly explained some of the test components but have not gone in depth with this.

15 Unit 15: Topological Sorts

** Reference Count topological sort implemented for Unit 15 **

15.1 Reference Count Sort Test

Figure 22: ReferenceCountSort Test

15.2 Self-Assessment

Grade: 3/5
Comments: I have provided full implementation for the ReferenceCountSort. The reason I deducted two points is becuase I have not made changes to the tests and is not Javadoc compliant.

16 Unit 16: Introduction to Concurrent Systems

Introductory unit, no work set but completed reading the notes.

17 Unit 17: Concurrent Systems and Java

17.1 Logbook Questions

Question 1: Will the test always terminate? I.e. is it certain that no matter how often you were to run the test it would always end in a finite length of time?
Answer: Yes, the test will always terminate once it reaches the end result. If it counts up, 10 will be the number it terminates on whereas when it counts down it will be 0.

Question 2: What is the shortest possible output for the test, in terms of the number of lines output?
Answer: 14 is the shortest possible output. This is when one of the counters finishes at the start and only requires the other to also be complete.

Question 3: What is the largest possible value that the count can reach when the test is run?
Answer: The highest possible value is 11. This occurs when the *up-counter* steps to 11 and finishes. This is due to the delay and not realising it has successfully finished because of the switch tracing.

Question 4: What is the lowest possible value that the count can reach when the test is run?
Answer: The lowest possible value is -1. This occurs when the *down* counter steps to -1 and finishes. Also due to the delay and not realising it has successfully finished because of the switch tracing.

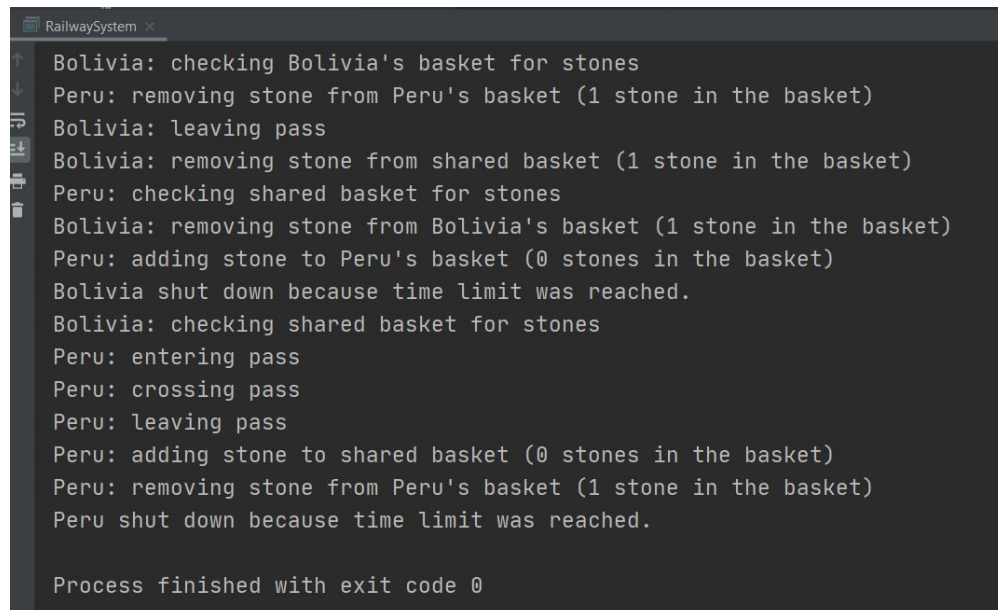## 17.2 Self-Assessment

Grade: 4/5
Comments: The questions have been answered and I have provided the explanation/reasoning for all of them. Despite this, I deducted a point because I could have explained the causes of the outputs more clearly.

18 Unit 18: Dekker's Algorithm

** Logbook Solution implemented for Unit 18 **

18.1 Railway System Output


```
RailwaySystem ×
 Bolivia: checking Bolivia's basket for stones
 Peru: removing stone from Peru's basket (1 stone in the basket)
 Bolivia: leaving pass
 Bolivia: removing stone from shared basket (1 stone in the basket)
 Peru: checking shared basket for stones
 Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
 Peru: adding stone to Peru's basket (0 stones in the basket)
 Bolivia shut down because time limit was reached.
 Bolivia: checking shared basket for stones
 Peru: entering pass
 Peru: crossing pass
 Peru: leaving pass
 Peru: adding stone to shared basket (0 stones in the basket)
 Peru: removing stone from Peru's basket (1 stone in the basket)
 Peru shut down because time limit was reached.

 Process finished with exit code 0
```

Figure 23: RailwaySystemTest Output

18.2 Extra Questions (Railway System)

Question 1a: The Bolivians complained that subversive timetables put together by the Peruvian railways could block the Bolivian trains for ever. Give a scenario, using the code from figure 4 on page 6, that shows how this could happen.
Answer: An example scenario could be where the Bolivian trains continuously see a stone in the basket even after their siesta. Moreover, the Peruvian railways are hogging the basket and not giving the Bolivians a chance to have their turn.

Question 1b: More seriously, one day the trains crashed. Give a scenario, using the code from figure 4 on page 6, that shows how this could happen.
Answer: A crash could occur when both drivers check the basket and it is empty and both of them don't check it the second time, so instead put a stone in. So the basket will have two stones where the drivers went to the basket but didn't check again. Now when they cross the pass a collision will occur.

Question 2a: Provide code for the Peru and Bolivia classes for this solution.
Answer:
Bolivia Code

```java
public void runTrain() throws RailwaySystemError {
        Clock clock = getRailwaySystem().getClock();
        while (!clock.timeOut()) {
                choochoo();
                while(!getSharedBasket().hasStone(this)) {
                        siesta();
                }
                crossPass();
                getSharedBasket().takeStone(this);
        }
}
```

Peru Code

```java
public void runTrain() throws RailwaySystemError {
        Clock clock = getRailwaySystem().getClock();
        while (!clock.timeOut()) {
                choochoo();
                while(getSharedBasket().hasStone(this)) {
                        siesta();
                }
                crossPass();
                getSharedBasket().putStone(this);
        }
}
```

Question 2b: Does this method prevent crashes?
Answer: Yes, because it relies on the other driver to do their tasks and doesn't allow two trains to proceed until the other driver completes their routine. Upon one driver completing their tasks, the other can do theirs. By the time one driver finishes their tasks, the pass will be cleared.

Question 2c: There was immediately a dispute between Peru and Bolivia about timetabling. Explain why.
Answer: Lets say a train from Bolivia crosses and takes the stone, and then another train from Bolivia comes to the entrance. This will result in the Bolivian driver to wait as there is no stone in the basket. Furthermore, this method only works on a 1-1 basis, and requires the pattern of 1 Bolivian train and 1 Peru train, hence many delays are created.

Question 3: Code proposal shown in figure 5 on page 7 (exercise). The trains crashed again, explain how?
Answer: The reason for why this causes the train to crash is simply because if both baskets are empty, it will cause both drivers to enter the pass - causing a collision.

Question 4a: Code for the attempted solution (stones in own basket before checking other driver's)
Answer:

```java
public void runTrain() throws RailwaySystemError {
        Clock clock = getRailwaySystem().getClock();
        while (!clock.timeOut()) {
                choochoo();
                getBasket().putStone(this);
                while(getSharedBasket().hasStone(this)) {
                        siesta();
                }
                crossPass();
                getBasket().takeStone(this);
        }
}
```

Question 4b: The trains stopped running, explain how.
Answer: An example scenario would be where both drivers put a stone in their own basket resulting in them being stuck in siesta. This is because both baskets will have a stone inside and not be able to proceed with the other methods.

Question 5a: Code for the solution (model exercise)
Answer (Model Exercise):

```java
public void runTrain() throws RailwaySystemError {
   Clock clock = getRailwaySystem().getClock();
   Railway nextRailway = getRailwaySystem().getNextRailway(this);
   while (!clock.timeOut()) {
      choochoo();
      getBasket().putStone(this);
      while (nextRailway.getBasket().hasStone(this)) {
         getBasket().takeStone(this);
         siesta();
         getBasket().putStone(this);
      }
      crossPass();
      getBasket().takeStone(this);
   }
}
```

Question 5b: The railways were still not happy. Why not?
Answer: This causes unnecessary delays as there is the chance that both driver's will siesta - causing a lot of time to be wasted.

Question 5c: Provide a scenario to demonstrate your answer to question 5b.
Answer: For example, if both drivers put a stone in their own basket then find a stone in the other drivers basket, they will go back and take the stone from their own basket and siesta. This can repeat until one driver finds the other basket to be empty (whilst they are in siesta).

18.3   Self-Assessment

Grade: 5/5
Comments: The logbook question has been implemented after learning about Dekker's algorithm. Also, I have completed most of the extra questions providing a very clear and concise explanation for each of them. For example, in some of the questions I gave an example scenario to answer the question clearly.

19 Unit 19: Semaphores

19.1 Logbook Questions

Question 1:
- After changing the order of criticalSection.vote() and noOfElements.vote() in the put method, this didn't cause an error and the program ran as it should. So the order of these isn't essential.
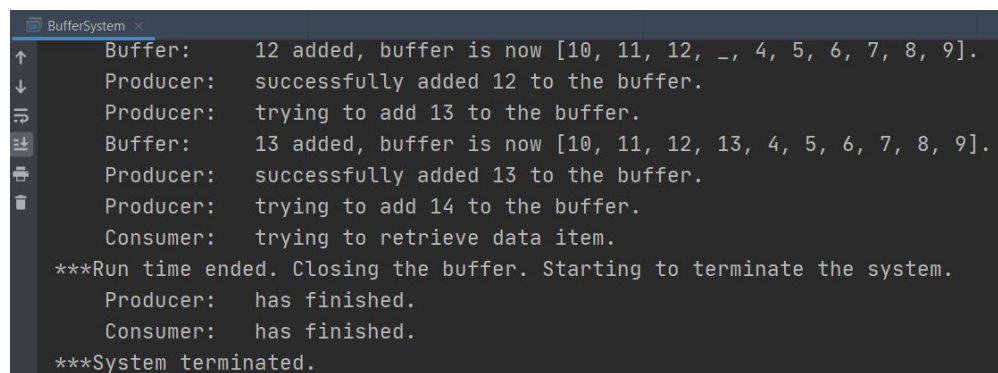On the other hand, the order of criticalSection.poll() and noOfElements.poll() within the get method is essential because if I swap these, the program runs into errors. For example, now 0 cannot be added to the buffer.

Question 2:
- The reason for the error in question 1 is because deadlock occurs, and it waits for one data item to be in the buffer. It eventually terminates after reaching the time limit.

Question 3:
- Order of the P() calls does matter, for example when the arguments are 10-20-fast-slow it eventually gets a total deadlock and items cannot be added or retrieved. This occurs when I swap the location of the two P() calls. Whereas if I kept them in their original state the program will run completely with no issues.


```
BufferSystem ×
     Buffer:     12 added, buffer is now [10, 11, 12, _, 4, 5, 6, 7, 8, 9].
     Producer:   successfully added 12 to the buffer.
     Producer:   trying to add 13 to the buffer.
     Buffer:     13 added, buffer is now [10, 11, 12, 13, 4, 5, 6, 7, 8, 9].
     Producer:   successfully added 13 to the buffer.
     Producer:   trying to add 14 to the buffer.
     Consumer:   trying to retrieve data item.
  ***Run time ended. Closing the buffer. Starting to terminate the system.
     Producer:   has finished.
     Consumer:   has finished.
  ***System terminated.
```

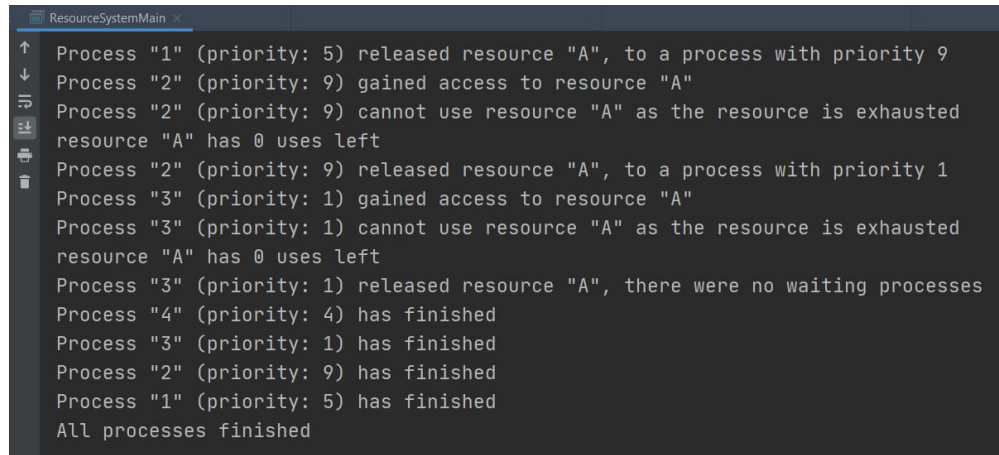Figure 24: Question 3 Deadlock

19.2 Self-Assessment

Grade: 4/5
Comments: I have given a detailed explanation for the three questions. The reason I deducted a point is because I could have gone into more depth as to why it actually occurs and possibly give an example scenario. This would help me get a clear understanding when I refer back to the notes.

20 Unit 20: Monitors (Lock and Conditions)

** LockResourceManager Implemented **

20.1 LockResourceManager Test

Using the code provided I tested the implementation and it successfully finishes all the processes using my implementation.

Figure 25: LockResourceManager Test

20.2 Self-Assessment

Grade: 4/5
Comments: The LockResourceManager implementation has been completed but I am lacking in documentation, therefore a point has been deducted.

21 Unit 24: Circuits and Maths

21.1 Logbook Question

Question: What is the matrix for the following circuit?

Figure 26: Circuit

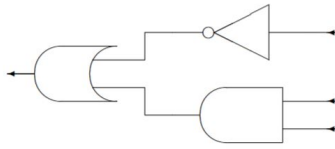**Answer:** This circuit uses the NOT, AND as well as the OR gate. To create the matrix, I will label these gates:

- NOT Gate = A input

- AND Gate = B input

- OR Gate = Z (OR output)

Now we can create a truth table and derive the matrix from it. My written result is below:

Figure 27: Matrix Answer

21.2 Extra Questions

* These can be found on the Unit 24 Matlab File or see Matlab Code*

21.2.1 Matlab Code

```matlab
a = 4 + 1i;
b = -2 + 6i;
i = 3*a %question i
ii = -4i*b %question ii
iii = a+b %question iii
iv = a-b %question iv
v = a*b %question v
vi = a/b %question vi
vii = abs(a) %question vii
viii = abs(b) %question viii
A = [
    0 -4  3  0
    1 2  0 -2
    0 -3  1 -1];
B = [
  1 -4  0  0  2
  0 -1  3 -2  0
  1  1  1  0 -1
  0  2 -3  3  4];
ix = 3*A %question ix
x = -5*B %question x
xi = A*B %question xi
xii = kron(A,B)  %question xii
C = [
  0     3+2i  2-4i
    2i -3+ 1i -4-3i];
xiii = transpose(C) %question xiii
xiv = conj(C) %question xiv
xv = transpose(conj(C)) %question xv
NOT = [
0 1
1 0];
FALSE = [1;0];
ZERO = FALSE;
TRUE = [0;1];
ONE = TRUE;
AND = [1,1,1,0;0,0,0,1];
xvi = kron(ONE,ONE) %question xvi
xvii = NOT*FALSE == ONE %question xvii
xviii = AND*kron(ONE,ONE) %question xviii
xviiiTwo = kron(ZERO,ONE) %question xviiiTwo
xix = [1,0,0,0;0,1,1,1]; %question xix
xx = NOT*xix %question xx
```

Listing 1: Extra Questions Matlab Code

21.3 Self-Assessment

Grade: 4/5
Comments: I have provided a full explanation of the derivation but have not provided testing or an alternative derivation. Hence, a point has been deducted to compensate for this.

22 Unit 27: Quantum Computing

22.1 Logbook Questions

Here is my written workout for the logbook question to show the states at each point:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}0\\1\end{bmatrix}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

$$B = H_V|0\rangle \; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$B = H_V|1\rangle \; \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$C = HB = H(H|1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0\\2\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

$$C = HB = H(H|0\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}2\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

Figure 28: Two Hadamard Gates

22.1.1 Logbook Explanation

As you can see when two hadamard gates are present the output will be the same as the base state.
For example $|1\rangle$ inputted will come back to $|1\rangle$, similarly with $|0\rangle$. Moreover, there will be a 50/50 chance to output either $|1\rangle$ or $|0\rangle$.

"If we began with the $|0\rangle$ state, we get constructive interference that builds us a new $|0\rangle$ state, and destructive interference that eliminates the $|1\rangle$ state." (Keio University, n.d.)
Similarly, starting with $|1\rangle$ will construct a new $|1\rangle$ and eliminates the $|0\rangle$ state.

The reason we cannot use the probability model is due to the lack of information and no knowledge of the vectors we are working with. Hence, getting to a result will be impossible.

22.2 Self-Assessment

Grade: 3/5
Comments: I have practically repeated the process for question 1(b) but added more detail into the explanation. Also, explained that the base value is returned.

23 Full Self-Assessment

Here is a list of the points I have accumulated from the unit tasks:

- Part 1: 15/15   unit 8 discounted

- Part 2: 15/15   unit 12 discounted

- Part 3: 13/15   unit 15 discounted

- Part 4: 12/15   unit 27 discounted
  Total : 55/60

References

Datta, S. (2020, October). Computing bubble sort time complexity. *Baeldung*. Retrieved from `https://www.baeldung.com/cs/bubble-sort-time-complexity`

Keio University. (n.d.). Quantum computing: Reversible evolution. *Understanding Quantum Computers*. Retrieved from `https://www.futurelearn.com/info/courses/intro-to-quantum-computing/0/steps/31561`