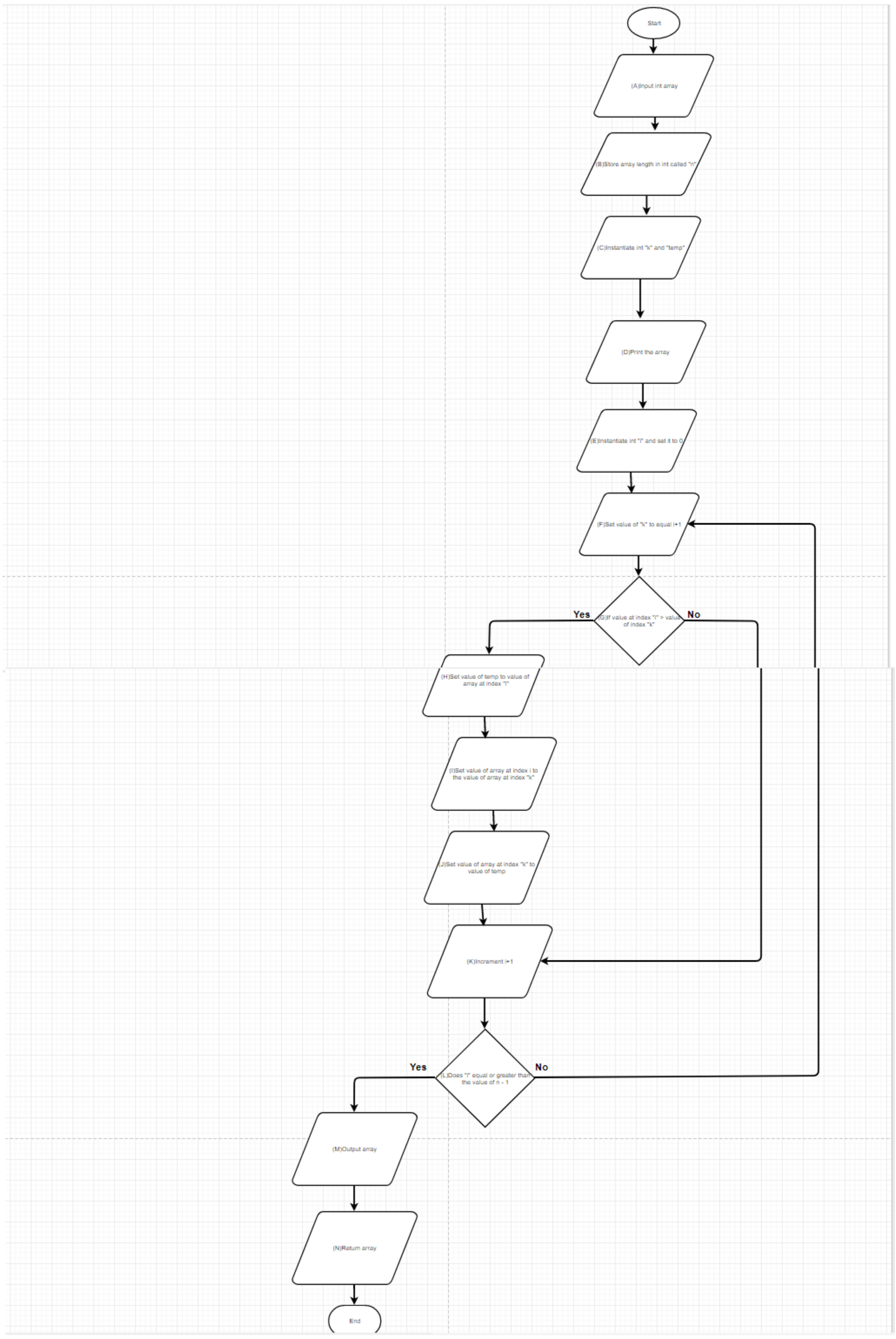
Software Engineering Assignment 2: Cieran Almond 1604959

1. **Code coverage test**

(i) Bubble sort flow chart


Start
(A)Input int array
(B)Store array length in int called "n"
(C)Instantiate int "k" and "temp"
(D)Print the array
(E)Instantiate int "i" and set it to 0
(F)Set value of "k" to equal i+1
(G)If value at index "i" > value of index "k"
Yes
No
(H)Set value of temp to value of array at index "i"
(I)Set value of array at index i to the value of array at index "k"
(J)Set value of array at index "k" to value of temp
(K)Increment i+1
(L)Does "i" equal or greater than the value of n - 1
Yes
No
(M)Output array
(N)Return array
End

(ii)

Path 1: Entire Program (A→B→C→D→E→F→G→H→I→J→K→L→M→N)

Path 2: Index “i” isn’t value of index “k” (A→B→C→D→E→F→G→K)

Path 3: “i” isn’t equal or greater than n-1 (A→B→C→D→E→F→G→H→I→J→K→L→F)

(iii)

| **Path** | **Test Case** | **Expected Result** |
|---|---|---|
| Full array to be sorted | [5, 1, 12, -5, 16] | [-5, 1, 5, 12,16] |
| Array with one element | [5] | Return one element[5] |
| Empty array | [] | Return empty array [] |

(iv) Additional test cases could be to count the number of iterations through the loop it takes to sort the list, and to print the number of iterations it took for it to be sorted. Best case being the list is already sorted, for example:

- **Worst case performance** *$O(n^2)$*
- **Best case performance** *$O(n)$*
- **Average case performance** *$O(n^2)$*
- **Worst case space complexity** *$O(n)$ total, $O(1)$ auxiliary*

Another test case could be sorting an already ordered list of values [-5, 1, 5, 12, 16] to see if it still follows the same path as an unordered array, or if it would take a new path.

Test case 3 – a list with all negative values of type int, an example array would look like so: [-5, -1, -6, -12, -15] to see if it could potentially take more time to sort, or because -15 is lower than -1 for example, see if it sorts the array as [-15, -12, -6, -5, -1].

Test case 4 - seeing if it would handle different data types, such as type double, an example array would look like so: [-2.3, 1.6. 7.8. 12.5. 15.6]. Though the expected result would be for the program to throw an error, as the array is only of type int, it will just prove that only type int is accepted in the method.

## 2. Unit Test – Present in java files

## 3. Design

(i)

Have I applied the software design principles of?
Abstraction:
I have used abstraction through the creation of my flow chart, which simplifies the bubble sort program, and only focuses on the core elements of the bubble sort which is the actual finished out come – the sorted list. Other characteristics, such as the TestDriver and TestOracle were not included as these are specific details that can be added later, and don't effect the core function of the program, and can be added later.

Modularity:
My application is modular in that sense that it applies Object-oriented techniques, and is split into 3 different classes (BubbleSort, TestDriver, TestOracle) which separates the program into several smaller modules. My program also uses further modulation inside these classes, in that declared "items" in the program are their own objects. An example of this is the declaration of my test arrays in their own methods, the methods being modularity from the classes, and the objects inside being modularity from the methods.

Loose Coupling or cohesion:
My program has "low coupling" since when the test oracle is called, it only needs to know the size of the array from the test driver, and not any other parameters. It also has "high cohesion" in that all the elements inside the program are relevant and all relate to the BubbleSort program; all 3 classes and their methods are necessary.

(ii)
Have you employed the object-oriented design mechanisms of?
Inheritance:
The BubbleSort and TestOracle programs both inherits properties of existing classes, that being the TestDriver, specifically the test array and outcome. These are compared to the length of the array in the TestOracle and what the expected outcome is. The bubble sort method takes the arrays from the TestDriver, prints the unsorted array, then runs through the method and prints the sorted array.

Encapsulation:
I used encapsulation for my variables declared in the bubble sort test driver, this was done by declaring them as private, meaning they can only be accessed by methods in their current class, and so are not accessible in the BubbleSort or TestOracle. I did this because these variables are only counts, and their only purpose is to be written to so I can calculate POFOD.

Information hiding:
The BubbleSort, TestOracle and TestDriver both do and don't make use of information hiding techniques; my variables "POFOD, count and successCount" all are declared private and so are hidden from outside classes, and all information relevant to those variables is stored within them. However the information within these variables can be manipulated by outside objects and methods, and so breaks the rule of "the information should only be able to be manipulated from itself".

Polymorphism:
My program doesn't make use of polymorphism techniques; I don't make use of any method inheritance or parent classes referencing child classes. I also don't use any reference

variables, the arrays are type integer, however when I print the arrays, I cast Arrays.toString onto them so they print the values correctly as a String. Otherwise the arrays are printed as the computer sees them, which in our eyes is just jargon.