**Name: MINH DUC NGUYEN**

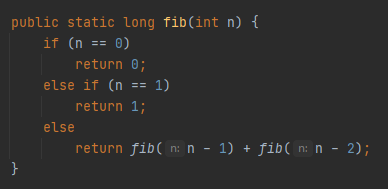
**CSC 143 Winter 2021**

**LAB 03: Algorithms**

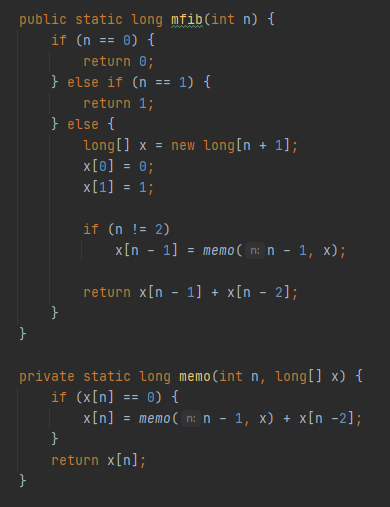
**RECURSIVE PROGRAMMING**

**Memoization:** The Fibonacci sequence

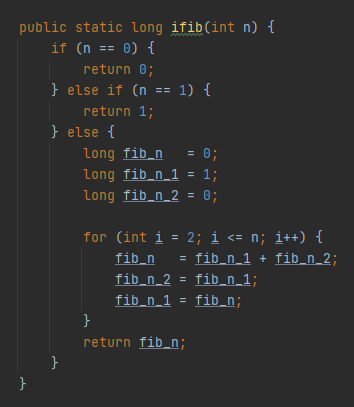
Update the **recursive method *fib(n)***to compute the Fibonacci value of n. This one is Fibonacci sequence solution in its most basic form.



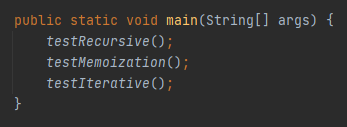
Write **a new version** of the Fibonacci method ***mfib(n)*** that is still recursive but is more efficient than the one in 1. There is a helper method ***memo(int n, long[] x)***. It accepts an additional parameter, the storage for the previous Fibonacci numbers, that we can carry through and modify during each recursive call.



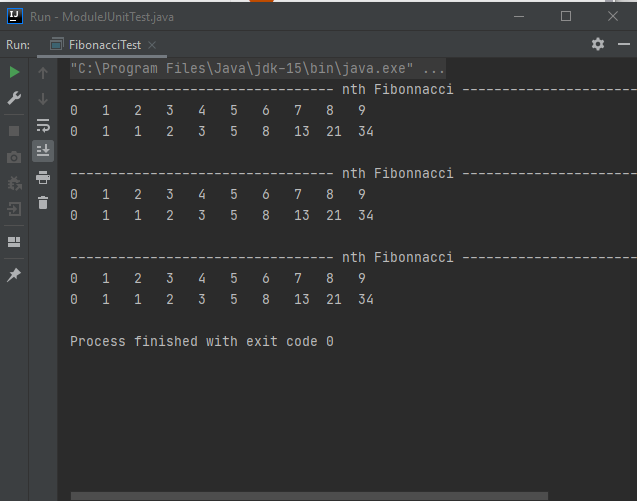
**A new version** of the Fibonacci method ***ifib(n)*** that uses iteration to generate the result for the nth value in the Fibonacci sequence.



**Main** method to test these above methods:

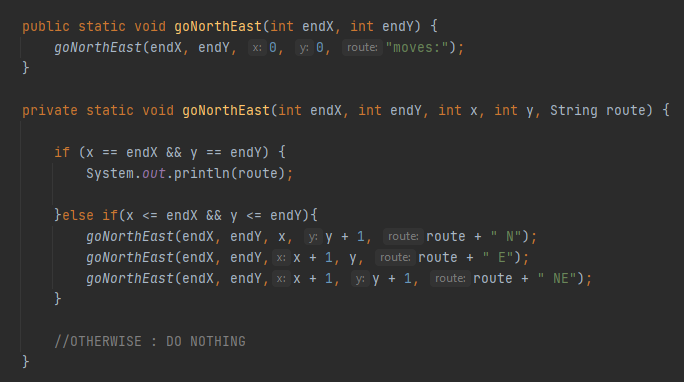


***OUTPUT:*** the results of three types of Fibonacci series

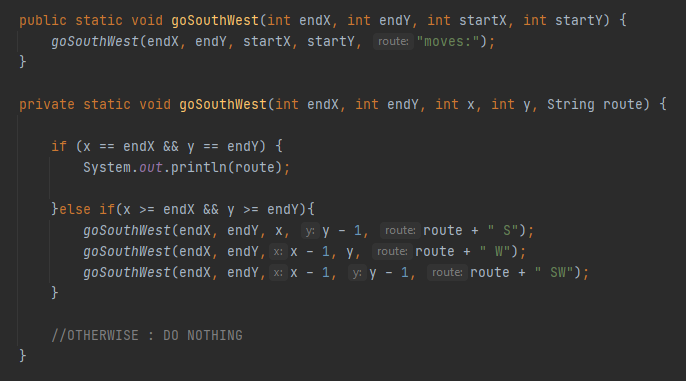


**Recursive Back Tracking**: travel South, West and South West and returns you back to the origin from your position at (2, 1).

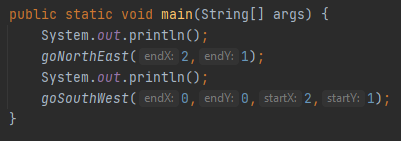
***goNorthEast():*** as the name of the method, it allows user to move in NE direction. There’s a private helper method with the same name, but with additional parameters.



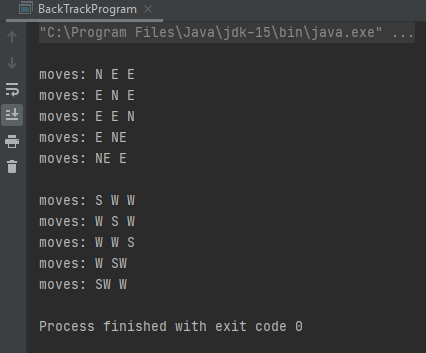
***goSouthWest():*** as the name of the method, it allows user to move in SE direction. There’s a private helper method with the same name, but with additional parameters.



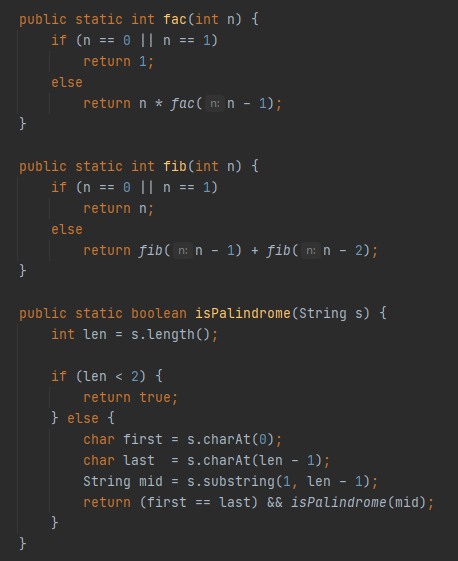
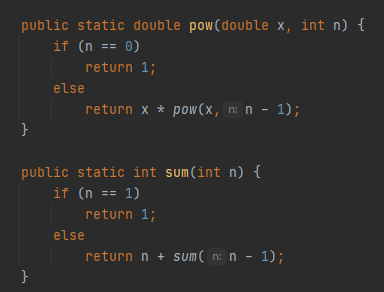
***Main()*** method to print out the solutions for (2, 1) and returning back to (0, 0).



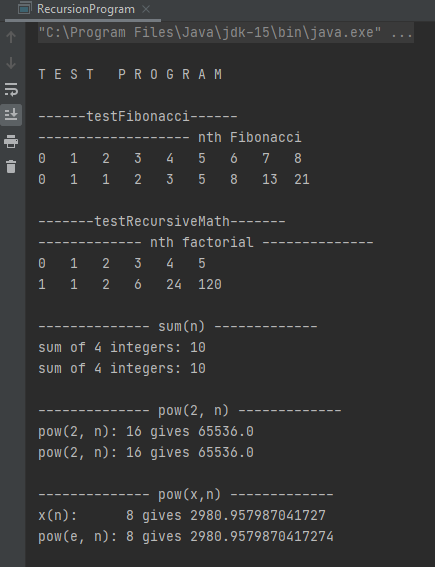
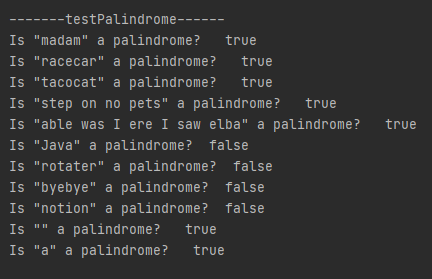
***OUTPUT:*** First, go to (2, 1), then back to (0, 0). Below is the direction for that.



**RecursionProgram test:** to create some recursive methods such as ***fac(), fib(), sum(), pow(),*** and ***isPalindrome()***

***OUTPUT:*** print out the test for above recursive methods

**EMPIRICAL ANALYSIS**

**PerformanceTest:**

Run the PerformanceTest and modify the runtime results in a new file named My\_Range\_RunTimes.xlsx (EXCEL file). Then we can compare it with the previous one.

***Questions:***

1. How similar or different is the performance of the data shown in the EXCEL file compared to your results?

My performanceTest’s results are larger but not too much than the given one. It means that it runs slower on my computer, because of the specs of different computers.

1. What do observe for each of the algorithms shown with their corresponding data set?

For the first algorithm, it takes a really long runtime to finish the test. So it looks like this algorithm is not really efficient, especially for large data input.

For the second algorithm, by somehow it is faster than the first one, but not too much. It still takes a huge amount of time when the number of input data is large.

For the third algorithm, it is the fastest one, even if the amount of input is much larger than the previous two. So this algorithm is the most efficient one among three I would say.

1. Can you tell which algorithm was the most efficient?

The third one is the most efficient algorithm, as mentioned above.

1. For algorithms 1 and 2, did reducing the amount of computations by half improve the runtime significantly? Explain your reasoning, if you felt it had a small or large change.

For these two algorithms, nested loops are used to examine every pair of elements in the array. So the runtime complexity in these scenarios are kind of O(n2). But the difference here is in algorithm 2, it disregards one of the (i, j) and (j, i) pairs, that give identical comparisons. So it can reduce the number of computations compared to algorithm 1. However, if we reduce the amount of computations by half, I think the runtime cannot be reduced significantly, because it is still the nested loop.

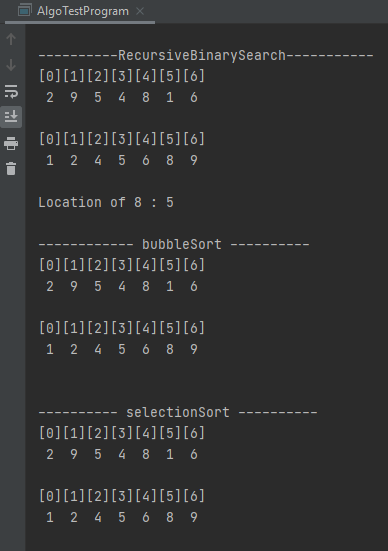
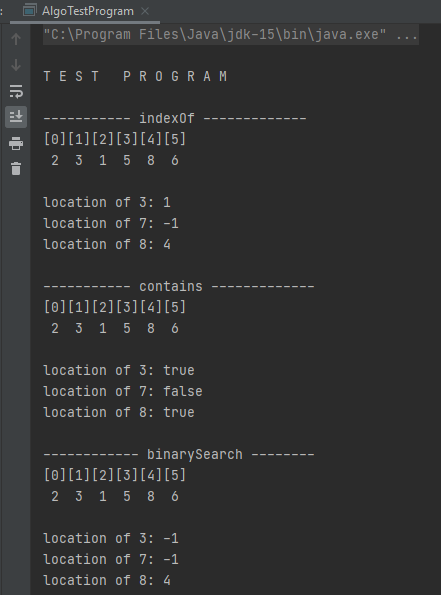
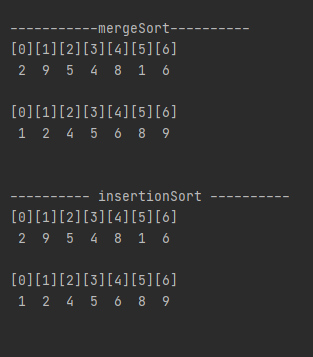
1. For algorithms 2 and 3, did reducing the number of loops improve the runtime significantly? Explain your reasoning, if you felt it had a small or large change.

For algorithm 3, it uses a loop to find the largest value and smallest value in the array, compute their difference and return this difference. Therefore, it looks like the runtime complexity in this case is just O(n), it is better than O(n2) of algorithms 1 and 2. So if we reduce the number of loops, I think it will improve the runtime significantly. It can be proved in runtime results in the Excel file mentioned above. We can see that the last algorithm works much efficiently compared to other ones, especially when the amount of input data is increasing.

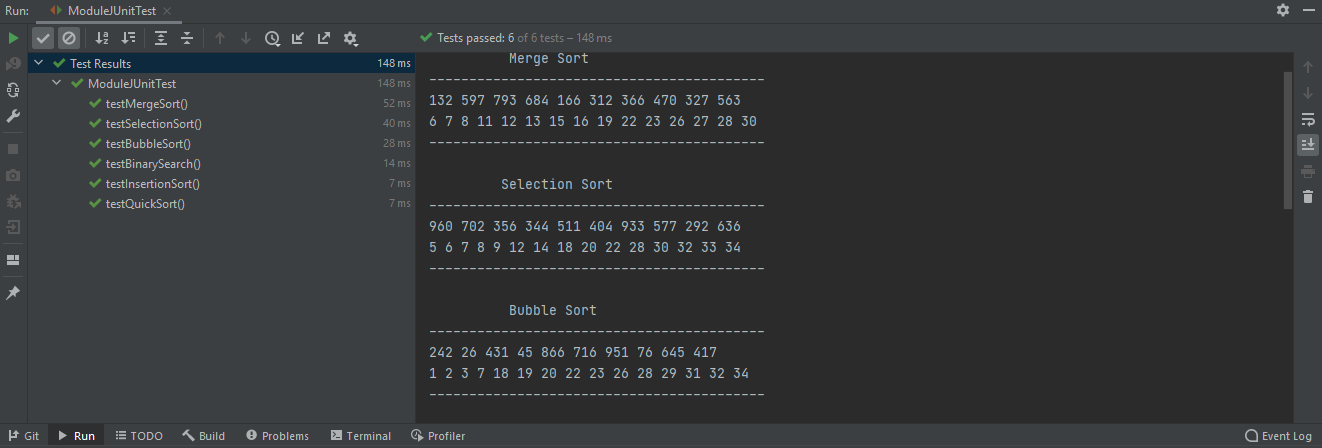
**SEARCHING / SORTING ALGORITHMS**

**AlgoTestProgram:** to test all the sorting and searching methods created in Module and RModule.

**OUTPUT of AlgoTestProgram**

** **

***OUTPUT of JUnitTest:*** After creating all required sorting and searching algorithms, I run the ModuleJUnitTest. It passes all the tests.

****