

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.

```
public static long fib(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

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.

```
public static long mfib(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        long[] x = new long[n + 1];
        x[0] = 0;
        x[1] = 1;

        if (n != 2)
            x[n - 1] = memo(n - 1, x);

        return x[n - 1] + x[n - 2];
    }
}

private static long memo(int n, long[] x) {
    if (x[n] == 0) {
        x[n] = memo(n - 1, x) + x[n - 2];
    }
    return x[n];
}
```

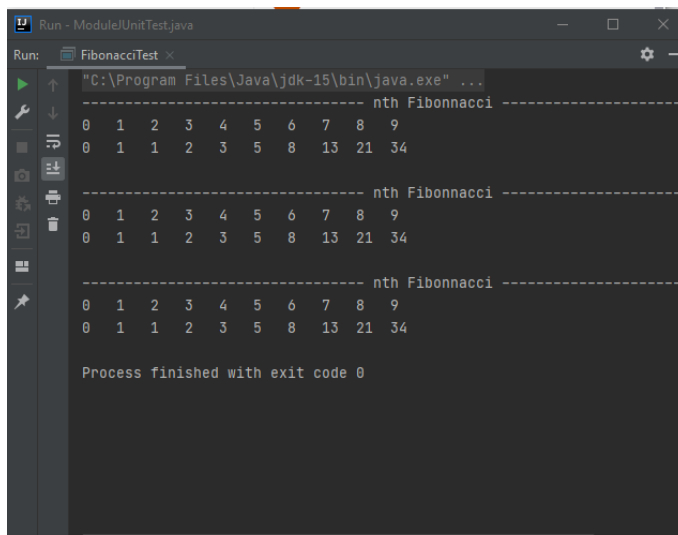
A new version of the Fibonacci method *ifib(n)* that uses iteration to generate the result for the  $n^{\text{th}}$  value in the Fibonacci sequence.

```
public static long ifib(int n) {  
    if (n == 0) {  
        return 0;  
    } else if (n == 1) {  
        return 1;  
    } else {  
        long fib_n = 0;  
        long fib_n_1 = 1;  
        long fib_n_2 = 0;  
  
        for (int i = 2; i <= n; i++) {  
            fib_n = fib_n_1 + fib_n_2;  
            fib_n_2 = fib_n_1;  
            fib_n_1 = fib_n;  
        }  
        return fib_n;  
    }  
}
```

Main method to test these above methods:

```
public static void main(String[] args) {  
    testRecursive();  
    testMemoization();  
    testIterative();  
}
```

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



```
Run - Module/UnitTest.java  
Run: FibonacciTest  
"C:\Program Files\Java\jdk-15\bin\java.exe" ...  
----- nth Fibonacci -----  
0 1 2 3 4 5 6 7 8 9  
0 1 1 2 3 5 8 13 21 34  
  
----- nth Fibonacci -----  
0 1 2 3 4 5 6 7 8 9  
0 1 1 2 3 5 8 13 21 34  
  
----- nth Fibonacci -----  
0 1 2 3 4 5 6 7 8 9  
0 1 1 2 3 5 8 13 21 34  
  
Process finished with exit code 0
```

**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.

```
public static void goNorthEast(int endX, int endY) {
    goNorthEast(endX, endY, x: 0, y: 0, route: "moves:");
}

private static void goNorthEast(int endX, int endY, int x, int y, String route) {

    if (x == endX && y == endY) {
        System.out.println(route);
    }else if(x <= endX && y <= endY){
        goNorthEast(endX, endY, x, y: y + 1, route: route + " N");
        goNorthEast(endX, endY, x: x + 1, y, route: route + " E");
        goNorthEast(endX, endY, x: x + 1, y: y + 1, route: route + " NE");
    }

    //OTHERWISE : DO NOTHING
}
```

**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.

```
public static void goSouthWest(int endX, int endY, int startX, int startY) {
    goSouthWest(endX, endY, startX, startY, route: "moves:");
}

private static void goSouthWest(int endX, int endY, int x, int y, String route) {

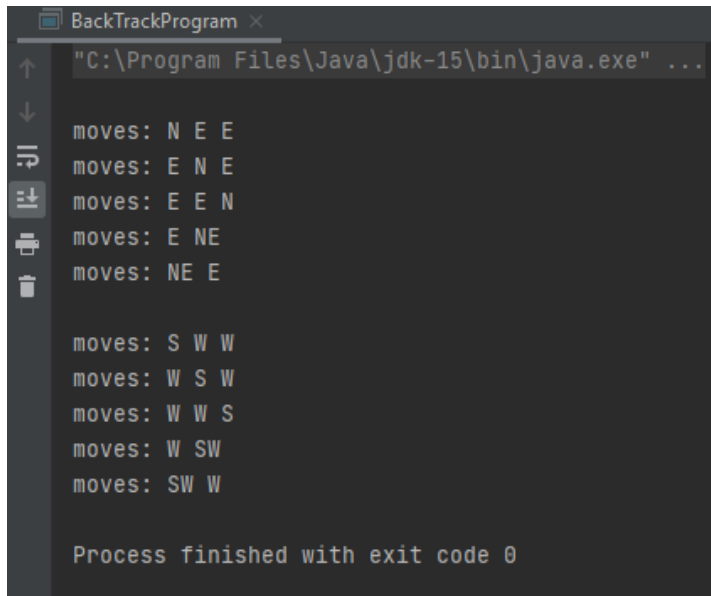
    if (x == endX && y == endY) {
        System.out.println(route);
    }else if(x >= endX && y >= endY){
        goSouthWest(endX, endY, x, y: y - 1, route: route + " S");
        goSouthWest(endX, endY, x: x - 1, y, route: route + " W");
        goSouthWest(endX, endY, x: x - 1, y: y - 1, route: route + " SW");
    }

    //OTHERWISE : DO NOTHING
}
```

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

```
public static void main(String[] args) {
    System.out.println();
    goNorthEast( endX: 2, endY: 1);
    System.out.println();
    goSouthWest( endX: 0, endY: 0, startX: 2, startY: 1);
}
```

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



```
BackTrackProgram
"C:\Program Files\Java\jdk-15\bin\java.exe" ...

moves: N E E
moves: E N E
moves: E E N
moves: E NE
moves: NE E

moves: S W W
moves: W S W
moves: W W S
moves: W SW
moves: SW W

Process finished with exit code 0
```

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

```
public static int fac(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * fac(n - 1);
}

public static int fib(int n) {
    if (n == 0 || n == 1)
        return n;
    else
        return fib(n - 1) + fib(n - 2);
}

public static boolean isPalindrome(String s) {
    int len = s.length();

    if (len < 2) {
        return true;
    } else {
        char first = s.charAt(0);
        char last = s.charAt(len - 1);
        String mid = s.substring(1, len - 1);
        return (first == last) && isPalindrome(mid);
    }
}
```

```
public static double pow(double x, int n) {
    if (n == 0)
        return 1;
    else
        return x * pow(x, n - 1);
}

public static int sum(int n) {
    if (n == 1)
        return 1;
    else
        return n + sum(n - 1);
}
```

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

```
RecursionProgram x
"C:\Program Files\Java\jdk-15\bin\java.exe" ...

T E S T   P R O G R A M

-----testFibonacci-----
----- nth Fibonacci
0  1  2  3  4  5  6  7  8
0  1  1  2  3  5  8  13 21

-----testRecursiveMath-----
----- nth factorial -----
0  1  2  3  4  5
1  1  2  6  24 120

----- sum(n) -----
sum of 4 integers: 10
sum of 4 integers: 10

----- pow(2, n) -----
pow(2, n): 16 gives 65536.0
pow(2, n): 16 gives 65536.0

----- pow(x,n) -----
x(n):      8 gives 2980.957987041727
pow(e, n): 8 gives 2980.9579870417274
```

```
-----testPalindrome-----
Is "madam" a palindrome?  true
Is "racecar" a palindrome?  true
Is "tacocat" a palindrome?  true
Is "step on no pets" a palindrome?  true
Is "able was I ere I saw elba" a palindrome?  true
Is "Java" a palindrome?  false
Is "rotater" a palindrome?  false
Is "byebye" a palindrome?  false
Is "notion" a palindrome?  false
Is "" a palindrome?  true
Is "a" a palindrome?  true
```

## **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.

2. 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.

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

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

4. 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  $O(n^2)$ . 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.

5. 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  $O(n^2)$  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

```
AlgoTestProgram
"C:\Program Files\Java\jdk-15\bin\java.exe" ...

TEST PROGRAM

----- indexOf -----
[0][1][2][3][4][5]
2 3 1 5 8 6

location of 3: 1
location of 7: -1
location of 8: 4

----- contains -----
[0][1][2][3][4][5]
2 3 1 5 8 6

location of 3: true
location of 7: false
location of 8: true

----- binarySearch -----
[0][1][2][3][4][5]
2 3 1 5 8 6

location of 3: -1
location of 7: -1
location of 8: 4

AlgoTestProgram
-----RecursiveBinarySearch-----
[0][1][2][3][4][5][6]
2 9 5 4 8 1 6

[0][1][2][3][4][5][6]
1 2 4 5 6 8 9

Location of 8 : 5

----- bubbleSort -----
[0][1][2][3][4][5][6]
2 9 5 4 8 1 6

[0][1][2][3][4][5][6]
1 2 4 5 6 8 9

----- selectionSort -----
[0][1][2][3][4][5][6]
2 9 5 4 8 1 6

[0][1][2][3][4][5][6]
1 2 4 5 6 8 9

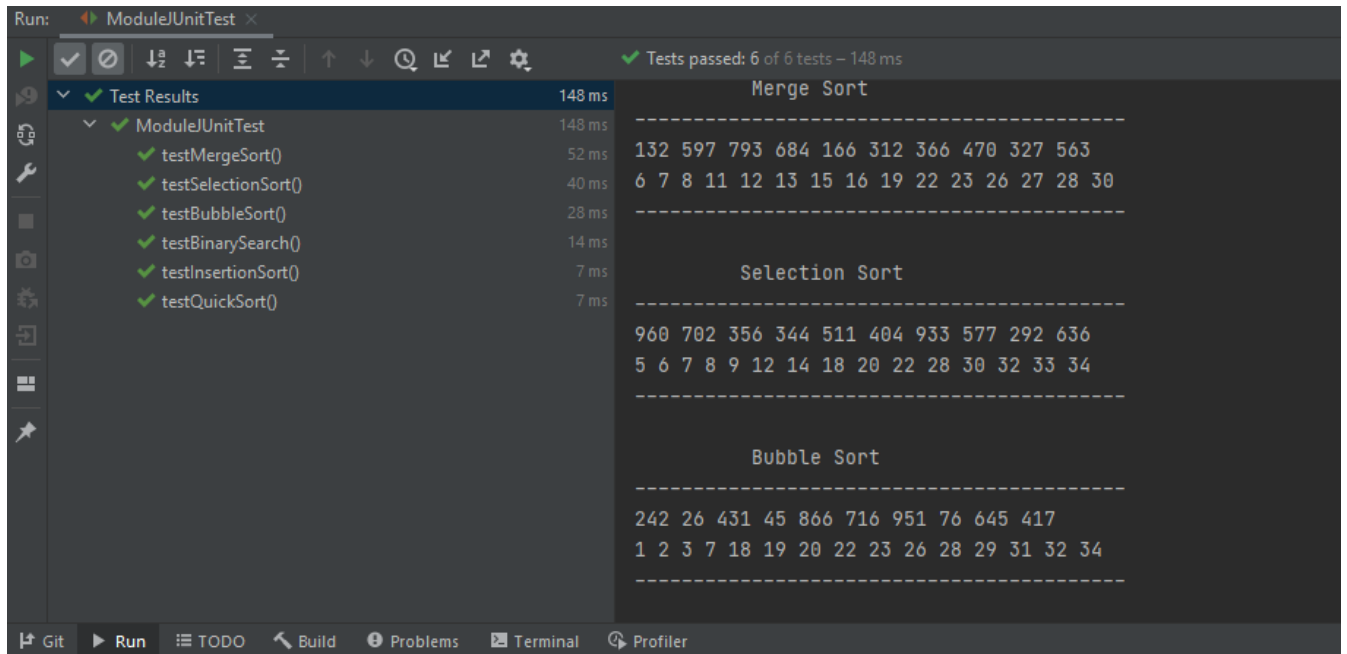
AlgoTestProgram
-----mergeSort-----
[0][1][2][3][4][5][6]
2 9 5 4 8 1 6

[0][1][2][3][4][5][6]
1 2 4 5 6 8 9

----- insertionSort -----
[0][1][2][3][4][5][6]
2 9 5 4 8 1 6

[0][1][2][3][4][5][6]
1 2 4 5 6 8 9
```

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



The screenshot shows the 'Run' window of an IDE, specifically the 'ModuleJUnitTest' tab. The window is divided into three main sections: a toolbar at the top, a 'Test Results' tree on the left, and a text output area on the right. The toolbar includes icons for running, stopping, and debugging, along with a status bar indicating 'Tests passed: 6 of 6 tests - 148 ms'. The 'Test Results' tree on the left shows a hierarchy starting with 'Test Results' (148 ms), which contains 'ModuleJUnitTest' (148 ms). Under 'ModuleJUnitTest', there are six test methods, all marked with green checkmarks: 'testMergeSort()' (52 ms), 'testSelectionSort()' (40 ms), 'testBubbleSort()' (28 ms), 'testBinarySearch()' (14 ms), 'testInsertionSort()' (7 ms), and 'testQuickSort()' (7 ms). The text output area on the right displays the results of these tests. It shows the title 'Merge Sort' followed by two lines of numbers: '132 597 793 684 166 312 366 470 327 563' and '6 7 8 11 12 13 15 16 19 22 23 26 27 28 30'. Below this, it shows the title 'Selection Sort' followed by two lines of numbers: '960 702 356 344 511 404 933 577 292 636' and '5 6 7 8 9 12 14 18 20 22 28 30 32 33 34'. Finally, it shows the title 'Bubble Sort' followed by two lines of numbers: '242 26 431 45 866 716 951 76 645 417' and '1 2 3 7 18 19 20 22 23 26 28 29 31 32 34'. The bottom of the window features a tabbed interface with 'Run' selected, and other tabs for 'Git', 'TODO', 'Build', 'Problems', 'Terminal', and 'Profiler'.

```
Run: ModuleJUnitTest x
[Icons] Tests passed: 6 of 6 tests - 148 ms

Test Results 148 ms
└─ ModuleJUnitTest 148 ms
    └─ testMergeSort() 52 ms
    └─ testSelectionSort() 40 ms
    └─ testBubbleSort() 28 ms
    └─ testBinarySearch() 14 ms
    └─ testInsertionSort() 7 ms
    └─ testQuickSort() 7 ms

Merge Sort
-----
132 597 793 684 166 312 366 470 327 563
6 7 8 11 12 13 15 16 19 22 23 26 27 28 30
-----

Selection Sort
-----
960 702 356 344 511 404 933 577 292 636
5 6 7 8 9 12 14 18 20 22 28 30 32 33 34
-----

Bubble Sort
-----
242 26 431 45 866 716 951 76 645 417
1 2 3 7 18 19 20 22 23 26 28 29 31 32 34
-----

Git Run TODO Build Problems Terminal Profiler
```