Laboratory practice No. 1: Recursion

Nicolás Restrepo López
Universidad Eafit
Medellín, Colombia
nrestrepol@eafit.edu.co

Mateo Restrepo Sierra
Universidad Eafit
Medellín, Colombia
mrestrepos@eafit.edu.co

3) Practice for final project defense presentation

1.1 Measure the times for recursive Array Sum, recursive Array Maximum and recursive Fibonacci series.

See annex: Laboratorio1.

2.1 Solve at least 5 exercises from Recursion 1 in CodingBat: http://codingbat.com/java/Recursion-1.

These exercises were solved in CodingBat account mateo.restrepo20171g062@hotmail.com*. See annex: Recursion1.*

2.2 Solve at least 5 exercises from Recursion 2 in CodingBat: http://codingbat.com/java/Recursion-2.

These exercises were solved in CodingBat account mateo.restrepo20171g062@hotmail.com*. See annex: Recursion2.*

2.3 Explain in your own words how does exercise GroupSum5 works.

The code must be something like this:

```
public boolean groupSum5(int start, int [] nums, int target) {
   if (start >= nums.length)
      return target == 0;
   if (nums[start] % 5 == 0){
      if (start < nums.length - 1 && nums[start + 1] == 1)
         return groupSum5(start + 2, nums, target - nums[start]);
      return groupSum5(start + 1, nums, target - nums[start]);
   }
   return groupSum5(start + 1, nums, target - nums[start])
       || groupSum5(start + 1, nums, target);
}
```

The exercise is a modified version of the popular Group Sum algorithm. This time, we must include every multiple of 5 from the array in the group, but if there is a number 1 following a 5, it must not be considered. Now, we will explain the algorithm.

The conditional if (line 2) gives the stop condition, in which *start* is larger than the size of the array. In this case, it must be returned if *target* is equal to zero (line 3).

What follows is the condition for multiples of 5 in the array to be chosen (line 4). But we also must consider the case of a number 1 following a 5, so we establish that we must skip that. To do that, we ask if our current *start* parameter is located before the last element on the array, and if the next element is 1 (line 5). If so, we change our parameters by adding 2 to our current *start* so it will not become an infinite loop (see that we add 2 instead of 1, otherwise we would include the 1 following the multiple of 5), and we subtract our current element on the array from the original *target*, which could be translated in that the element does count for the group (line 6).

Next, we have the return statement for the case in which the current element is a multiple of 5 not followed by a 1. If that happens, we change our parameters similarly to the previous case, but we add 1 to our *start* instead of 2 because there is no reason to skip the next element (line 7).

Finally, if our current element is not a multiple of 5, we proceed by splitting our algorithm into two cases: the first one (line 8) choses our current element by using the same return statement than in line 7. The second case (line 9) does add 1 to our *start* parameter to avoid entering an infinite loop, but does not subtract our current element, so it is not chosen for the group.

2.4 Calculate the complexity for the Online Exercises in 2.1 and 2.2. Then, add it to the PDF report.

Factorial (Recursion 1)

```
public int factorial(int n) {
  if(n == 1)                                  //C1
     return 1;                                //C2
  return n * factorial(n - 1);                //C3 + T(n-1)
}
```

T(n) = C1 + C3 + T(n – 1)
T(n) = C + T(n – 1)
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), by Sum Law.
T(n) = O (n), by Product Law.

BunnyEars (Recursion 1)

```
public int bunnyEars(int bunnies) {
  if (bunnies == 0)                           //C1
    return 0;                                 //C2
  return 2 + bunnyEars(bunnies - 1);          //C3 + T(n-1)
}
```

T(n) = C1 + C3 + T(n – 1)
T(n) = C + T(n – 1)
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), by Sum Law.
T(n) = O (n), by Product Law.

Fibonacci (Recursion 1)

```
public int fibonacci(int n) {
   if (n <= 1)                                //C1
      return n;                               //C2
   return fibonacci(n - 2) + fibonacci(n - 1);   //C3 + T(n-1) + T(n-1)
}
```

T(n) = C1 + C3 + T (n – 1) + T (n – 1)
T(n) = C + 2*T (n – 1)
T(n) = C'*2^{n-1} + C*(2^n – 1)
T(n) = O (C'*2^{n-1} + C*(2^n – 1))
T(n) = O (C*(2^n – 1)), by Sum Law.
T(n) = O (2^n – 1), by Product Law.
T(n) = O (2^n), by Sum Law.

BunnyEars2 (Recursion 1)

```
public int bunnyEars2(int bunnies) {
   if (bunnies == 0)                          //C1
      return bunnies;                         //C2
   if (bunnies % 2 == 1)                      //C3
      return 2 + bunnyEars2(bunnies - 1);     //C4 + T(n-1)
   return 3 + bunnyEars2(bunnies - 1);        //C5 + T(n-1)
}
```

T(n) = C1 + C3 + C4 + T(n – 1)
T(n) = C + T(n – 1)
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), by Sum Law.
T(n) = O (n), by Product Law.

Array220 (Recursion 1)

```
public boolean array220(int[] nums, int index) {
   if (nums.length < 2 || index == nums.length - 1)  //C1
      return false;                                  //C2
   if (nums[index + 1] == nums[index] * 10)          //C3
      return true;                                   //C4
   return array220(nums, index + 1);                 //C5 + T(n-1)
}
```

T(n) = C1 + C3 + C5 + T(n – 1)
T(n) = C + T(n – 1)
T(n) = C' + C*n
T(n) = O (C' + C*n)
T(n) = O (C*n), by Sum Law.
T(n) = O (n), by Product Law.

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

GroupSum (Recursion 2)

```
public boolean groupSum(int start, int[] nums, int target) {
   if (start >= nums.length)                                  //C1
      return (target == 0);                                   //C2
   return groupSum(start + 1, nums, target - nums[start])     //C3 + T(n-1)
       || groupSum(start + 1, nums, target);                  //C4 + T(n-1)
}
```

T(n) = C1 + C3 + T (n – 1) + C4 + T (n – 1)
T(n) = C + 2*T (n – 1)
T(n) = C'*2^{n-1} + C*(2^n – 1)
T(n) = O (C'*2^{n-1} + C*(2^n – 1))
T(n) = O (C*(2^n – 1)), by Sum Law.
T(n) = O (2^n – 1), by Product Law.
T(n) = O (2^n), by Sum Law.

GroupSum6 (Recursion 2)

```
public boolean groupSum6(int start, int[] nums, int target) {
   if (start >= nums.length)                                  //C1
      return target == 0;                                     //C2
   if (nums[start] == 6)                                      //C3
      return groupSum6(start + 1, nums, target - nums[start]);
                                                              //C4 + T(n-1)
   return groupSum6(start + 1, nums, target - nums[start])    //C5 + T(n-1)
       || groupSum6(start + 1, nums, target);                 //C6 + T(n-1)
}
```

T(n) = C1 + C3 + C5 + T (n – 1) + C6 + T (n – 1)
T(n) = C + 2*T (n – 1)
T(n) = C'*2^{n-1} + C*(2^n – 1)
T(n) = O (C'*2^{n-1} + C*(2^n – 1))
T(n) = O (C*(2^n – 1)), by Sum Law.
T(n) = O (2^n – 1), by Product Law.
T(n) = O (2^n), by Sum Law.

GroupNoAdj (Recursion 2)

```
public boolean groupNoAdj(int start, int[] nums, int target) {
   if (start >= nums.length)                                  //C1
      return target == 0;                                     //C2
   return groupNoAdj(start + 2, nums, target - nums[start]) //C3 + T(n-1)
       || groupNoAdj(start + 1, nums, target);                //C4 + T(n-1)
}
```

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

T(n) = C1 + C3 + T (n – 1) + C4 + T (n – 1)
T(n) = C + 2*T (n – 1)
T(n) = C'*2^{n-1} + C*(2^n – 1)
T(n) = O (C'*2^{n-1} + C*(2^n – 1))
T(n) = O (C*(2^n – 1)), by Sum Law.
T(n) = O (2^n – 1), by Product Law.
T(n) = O (2^n), by Sum Law.

GroupSum5 (Recursion 2)

```
public boolean groupSum5(int start, int [] nums, int target) {
   if (start >= nums.length)                                    //C1
      return target == 0;                                       //C2
   if (nums[start] % 5 == 0){                                   //C3
      if (start < nums.length - 1 && nums[start + 1] == 1)      //C4
         return groupSum5(start + 2, nums, target - nums[start]);
                                                                //C5 + T(n-1)
      return groupSum5(start + 1, nums, target - nums[start]);//C6 + T(n-1)
   }
   return groupSum5(start + 1, nums, target - nums[start])      //C7 + T(n-1)
       || groupSum5(start + 1, nums, target);                   //C8 + T(n-1)
}
```

T(n) = C1 + C3 + C4 + C5 + T (n – 1) + C7 + T (n – 1) + C8 + T(n – 1)
T(n) = C + 3*T (n – 1)
T(n) = (1/6)*(3C*(3^n – 1) + 2C'*3^n)
T(n) = O ((1/6)*(3C*(3^n – 1) + 2C'*3^n))
T(n) = O (3C*(3^n – 1) + 2C'*3^n), by Product Law.
T(n) = O (3C*(3^n – 1)), by Sum Law.
T(n) = O (3^n – 1), by Product Law.
T(n) = O (3^n), by Sum Law.

GroupSumClump (Recursion 2)

```
public boolean groupSumClump(int start, int[] nums, int target) {
   if (start >= nums.length)                                  //C1
      return target == 0;                                     //C2
   int suma = nums[start];                                    //C3
   int count = 1;                                             //C4
   for (int i = start + 1; i < nums.length; i++){             //C5*n
      if (nums[i] == nums[start]){                            //C6*n
         suma += nums[i];                                     //C7*n
         count++;                                             //C8*n
      }
   }
   return groupSumClump(start + count, nums, target - suma)//C9 + T(n-1)
       || groupSumClump(start + count, nums, target);      //C10 + T(n-1)
}
```

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

$T(n) = C1 + C3 + C4 + C5*n + C6*n + C7*n + C8*n + C9 + T(n-1) + C10 + T(n-1)$
$T(n) = C + C'*n + 2*T(n-1)$
$T(n) = 2^{n-1}(4C + 2C' + C'') - C*n - 2C - C'$
$T(n) = O(2^{n-1}(4C + 2C' + C'') - C*n - 2C - C')$
$T(n) = O(2^{n-1}(4C + 2C' + C''))$, by Sum Law.
$T(n) = O(2^{n-1})$, by Product Law.
$T(n) = O(2^n)$

2.5 Explain in your own words variables m and n in the calculation for complexity in 2.4

In the calculation for complexity, there is only one variable, which is n and denotes the input parameter for the algorithm. The different C terms are just constants for already established times for making decisions, comparisons or returning a value.

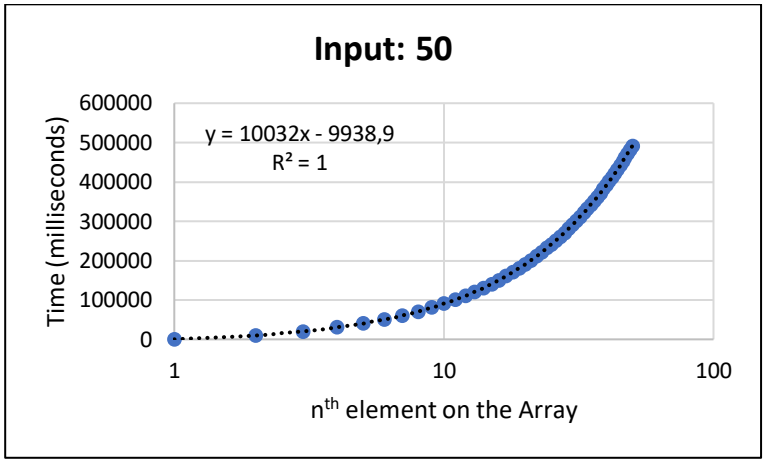
3.1 According to 1.1, complete the tables with times in milliseconds.

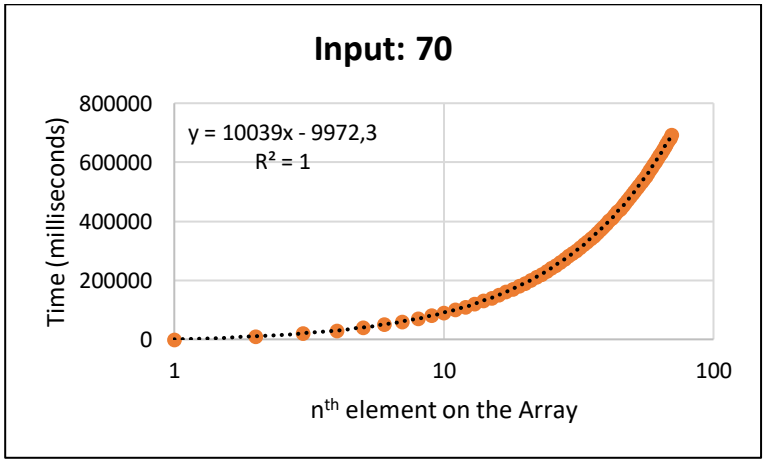
	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
ArraySum	1.003.989.979	10.039.989.979	100.399.989.979	1.003.999.989.979
ArrayMax	1.003.600.065	10.036.000.065	100.360.000.065	1.003.600.000.065
Fibonacci	$9{,}4 \times 10^{21813}$	$2{,}58 \times 10^{218106}$	$4{,}09 \times 10^{2181030}$	$4{,}03 \times 10^{21810272}$

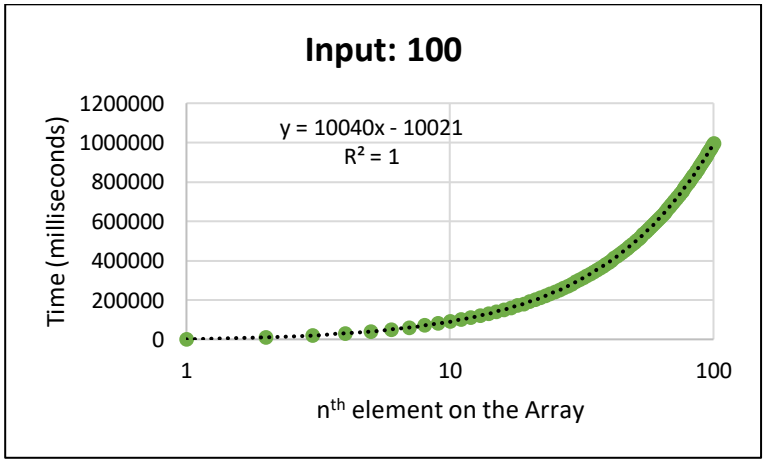
These values were obtained by implementing the equation for each graphic in 3.2 with the highest input (100 for ArraySum and ArrayMax, and 20 for Fibonacci). The reason is that it would take an enormous amount of time, and those values would surpass the range for integers and long type numbers in Java (2.147.483.647 and 9.223.372.036.854.775.808, respectively).

3.2 Graphic the time it took to execute Array Sum, Array Maximum and recursive Fibonacci, for different size inputs.

ArraySum

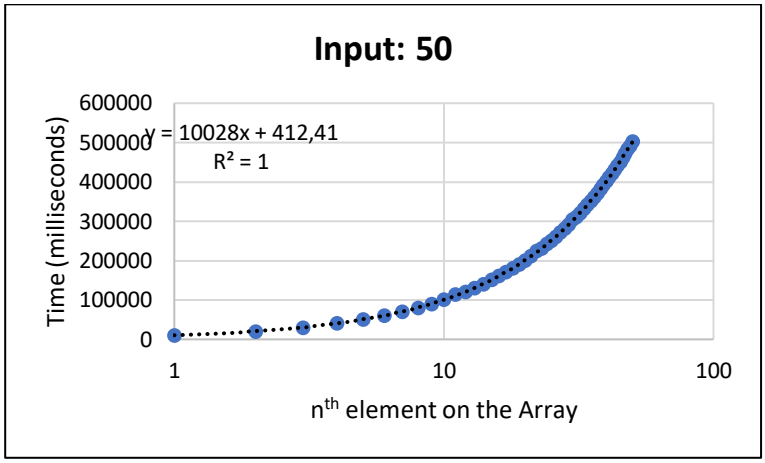


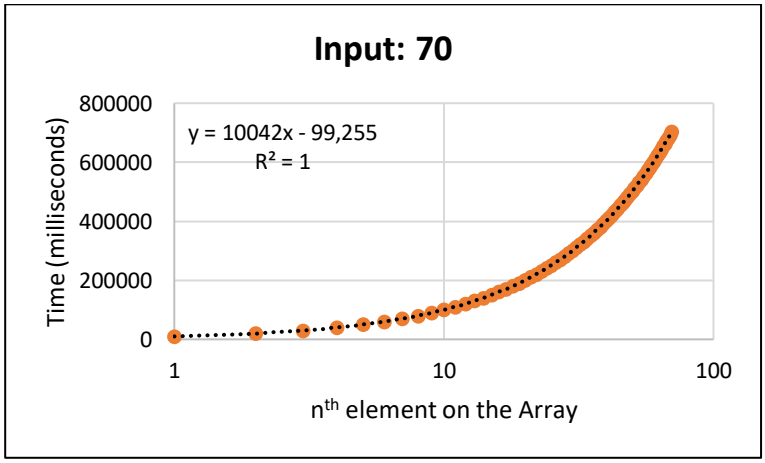


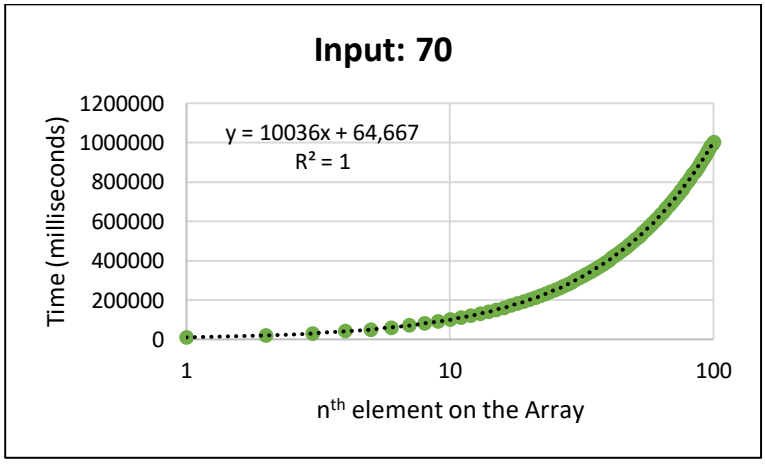


PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

ArrayMax

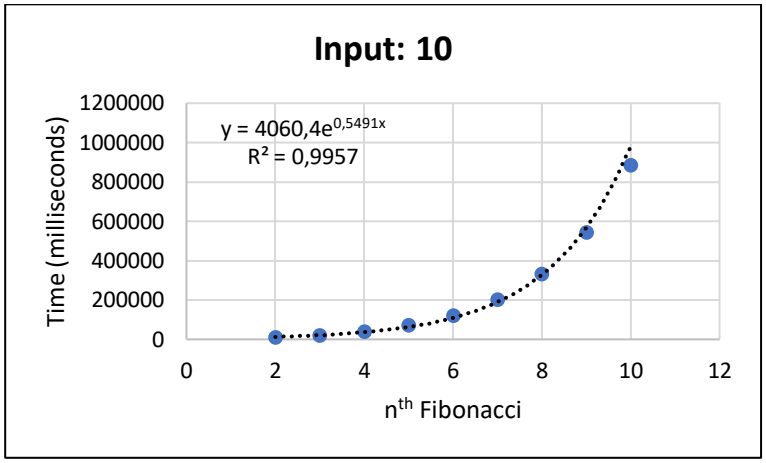


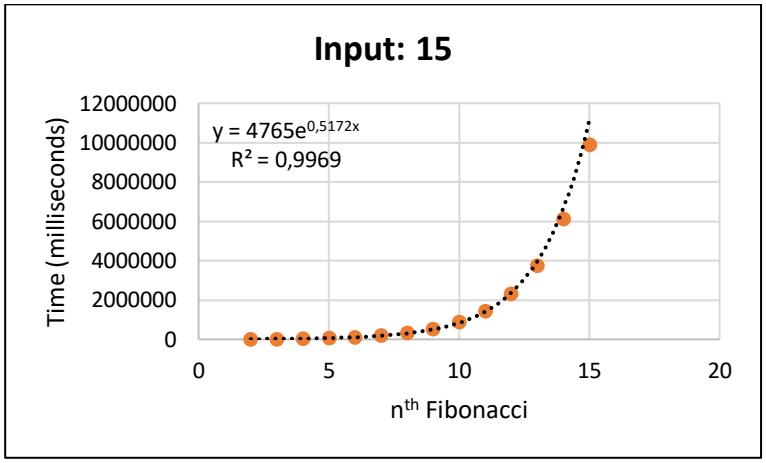


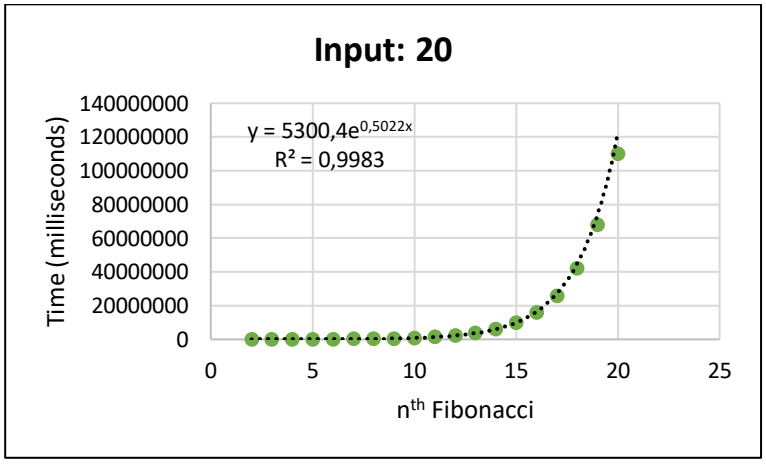


PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

Fibonacci







PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

3.3 What do you conclude for the obtained times in the laboratory and the theoretical results?

The times obtained in the laboratory have a similar behavior than the supposed complexity of each algorithm. Those complexities are:

- ArraySum: O (n)
- ArrayMax: O (n)
- Fibonacci: O (2^n)

When looking at R^2 values on each graphic, we can see that the resulting equations are very similar to theoretical values.

3.4 What did you learn about Stack Overflow?

Every computer has a maximum stack size. A stack overflow happens when certain process exceeds that size, and it is common for an algorithm with wrong recursive calls. In this case, the recursive method has a problem with the stop condition, whether it does not exist or it is not possible to reach it. Let's see an example:

We have a method with an integer as an input, and it returns its factorial.

```
public int factorial (int n) {
   if (n == 100)
      return n;
   return n*factorial(n-1);
}
```

We see that there is an error on line 2, because n would never be 100, not even when n is bigger than that because it would still return 100 as an answer. It is necessary to look for well-established stop conditions in recursive methods in a way it will not become an infinite loop, which is a common situation in which a Stack Overflow occurs.

3.5 Which is the biggest value that Fibonacci could calculate? Why? Why could not Fibonacci be executed for one million?

We will describe the behavior of Fibonacci when called from main with time sleep. The Fibonacci algorithm return a *long* type value, which has a range that goes from –9.223.372.036.854.775.807 to 9.223.372.036.854.775.807. If we take as a reference the third graphic in 3.2, we can see that the function that best suits the behavior of its complexity is $y = 5300{,}4e^{0{,}5022x}$, so we can use its

inverse to find the highest F(n) that could be calculated. The inverse of the function is y = (ln x – ln 5300,4) / 0,5022, so we replace x with the highest possible *long* value. That's how we get 69,87800639, and by approaching to a lower integer we have 69. Finally, F(69) would be the highest number the algorithm could calculate before surpassing the range for *long* values when implementing time sleep.

That being said, it could not be possible to calculate F(1'000.000) with our algorithm because it would surpass by and incredibly high difference the range for *long* values, having into account that we already calculated that number in 3.1: $2{,}58 \times 10^{218106}$. It is a 218.106-digit number.

3.6 How can Fibonacci be calculated for big values?

A possible solution might be the use of a technique called *Dynamic Programming*, which consists on storing the generated result by secondary problems. Why? Because this allows to make calculations in an easier way by turning unnecessary to repeat the same process over and over. However, there are two conditions that could say if DP can be used in a certain algorithm:

- Overlapping Sub-Problems: These are the problems that are to be solved repeatedly, so the idea is to store those answers to use them in the future.
- Optimal Substructure: This means that the problem can be solved by using the stored answers.

3.7 What do you conclude about the complexity for CodingBat exercises in Recursion1 respecting to Recursion 2?

What we can conclude about the complexity of algorithms in Recursion 1 is that their complexity is mostly T(n) = C + T(n - 1), except for Fibonacci. In O notation, this is equivalent to O(n), because it is necessary to make only one recursive call. On the other hand, complexity in Recursion 2 in O notation is $O(2^n)$, so it is different to the one in Recursion 1 and take longer times because they have more than one recursive call. There were multiple constants in some of the algorithms, but when making profs with big numbers, those constants do not make a huge difference on their complexity.

4) Practice for midterms

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

1. Pepito wrote an algorithm that, given an array of integers, decides if it is possible to choose a subset of those integers in a way the sum of those integers equals a target value. The start parameter works as a counter and represents an index on the array.

```
01  public boolean SumaGrupo(int start, int[] nums, int target) {
02    if (start >= nums.length) return target == 0;
03      return SumaGrupo(start + 1, nums, target - nums[start])
04           || SumaGrupo(____,___,___);
05  }
```

Which parameters would you write on the recursive call in line 4 to make the program work?

Start + 1, nums, target

2. Pepito wrote the following code using recursion:

```
private int b(int[ ] a, int x, int low, int high) {
  if (low > high)
    return -1;
  int mid = (low + high)/2;
  if (a[mid] == x)
    return mid;
  else if (a[mid] < x)
    return b(a, x, mid+1, high);
  else
    return b(a, x, low, mid-1);
}
```

Which recurrence equation describes the algorithm's behavior in the worst of cases?

a) $T(n)=T(n/2)+C$
b) $T(n)=2.T(n/2)+C$
c) $T(n)=2.T(n/2)+Cn$
d) $T(n)=T(n-1)+C$

3. Dayla and Kefo are back. This time they have brought an interesting game, in which Kefo first chooses a number n between 1 and 20, inclusive, and then he chooses three numbers a, b and c between 1 and 9, inclusive. After that, he gives those numbers to Dayla and she must say Kefo the maximum amount of

numbers taken from a, b and c (it can be taken more than once) that have as result n when summed.

As an example, if Kefo chooses n = 14, a = 3, b = 2 and c = 7, which possibilities are there to obtain 14 with a, b and c?

- 7+7 = 14. Quantity is 2.
- 7+3+2+2 = 14. Quantity is 4.
- 3+3+3+3+2 = 14. Quantity is 5.
  …
- 2+2+2+2+2+2+2 = 14. Quantity is 7.

The maximum amount of numbers is 7. This would be the answer Dayla gives Kefo. As Dayla is clever, she has designed an algorithm to determine the maximum amount of numbers and wants you to finish her code. Assume that there is at least one way to obtain n using a, b and c in different quantities, even if those numbers are summed zero times.

```
01 int solucionar (int n, int a, int b, int c) {
02    if (n == 0 || (n < a && n < b && n < c))
03       return 0;
04    int res = solucionar (___) + 1;
05    res = Math.max (___, ___);
06    res = Math.max (___, ___);
07    return res;
08 }
```

3.1 Fill in the blank in line 04.

n – a, a, b, c

3.2 Fill in the blank in line 05.

res, solucionar (n – b, c, a, c) + 1

3.3 Fill in the blank in line 06.

res, solucionar (n – c, b, c, a) + 1

4. What does the *desconocido* algorithm calculate and which is its asymptotic complexity in the worst of cases?

```
01 public int desconocido (int [] a) {
```

```
02      return aux(a, a.length-1);
03    }
04    public int aux(int[] a, int n){
05      if(n < 1) return a[n];
06      else return a[n] + aux(a, n-1);
07    }
```

a) The sum of the elements of the array and is
b) Orders the array and is O(n*log n).
c) The sum of the elements of the array and is O(1).
d) The maximum value of the array and is O(n).
e) The sum of the elements of the array and is O(n).

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co