Laboratory practice No. 1: Recursion

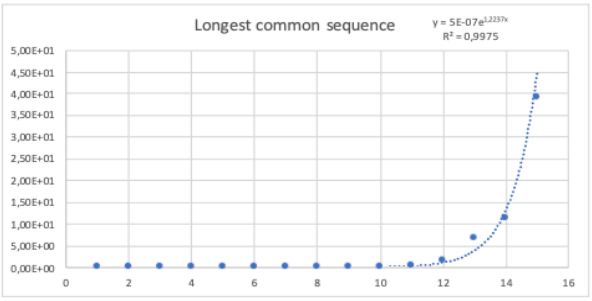
Vicente Aristizabal Ospina
Universidad Eafit
Medellín, Colombia
Varisti7@eafit.edu.co

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3) Practice for final project defense presentation

3.1, 3.2, 3.3:

Size (n)	Time (s)
1	2,86E-06
2	5,01E-06
3	1,72E-05
4	5,72E-05
5	0,000192165
6	0,000741959
7	0,002493858
8	0,005534887
9	0,026319981
10	0,103884935
11	0,323891878
12	1,632820845
13	6,674180031
14	11,23176384
15	38,95291305

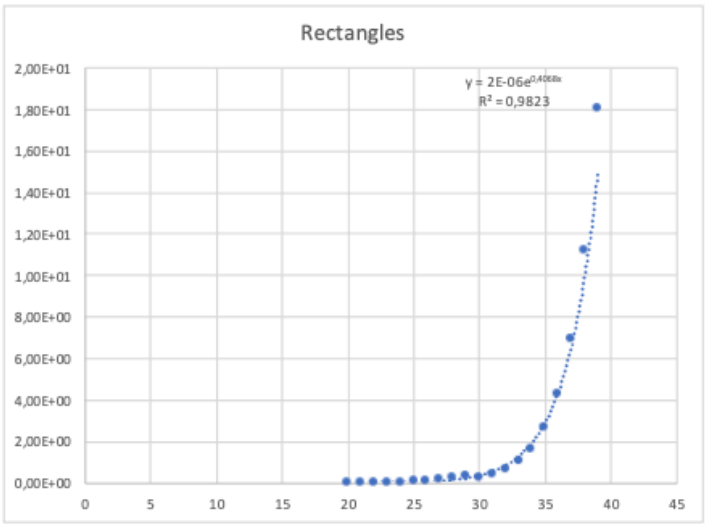


Common sequence by complexity	
Tamaño (n)	300
Tamaño total	1,E+91
Segundos (Tt/2GHz)	6,111107929003 46E+81
Horas	1,697530E+78

3.3) Taking in account the algorithm's complexity, this WOULDN'T be the one for taking this problem, because the number of instructions to solve it raises at 2^n, so for a 300.000 size of (n) it would take 2^300.000 instructions. Not even the computer is able to calculate the number. In yellow is an example of 2^300.

$$T(n) = (2^n - 1)(c1 + c3) + c_1\ 2^(n - 1)$$

Size (n)	Time (s)
20	6,20E-03
21	4,55E-03
22	1,29E-02
23	2,36E-02
24	0,04935503
25	0,079976082
26	0,100703001
27	0,176604033
28	0,24488306
29	0,347311974
30	0,278718948
31	0,402431965
32	0,634913206
33	1,017865181
34	1,632277012
35	2,63162899
36	4,297087908
37	6,946293831
38	11,17777205
39	18,06480503



Common sequence by complexity	
Tamaño (n)	300
Tamaño total	1,E+91
Segundos (Tt/2GHz)	6,111107929003 46E+81
Horas	1,697530E+78

3.3) Taking in account the algorithm's complexity, this WOULDN'T be the one for taking this problem, because the number of instructions to solve it raises at 2^n, so for a 300.000 size of (n) it would take 2^300.000 instructions. Not even the computer is able to calculate the number. In yellow is an example of

$$T(n) = C1 + T(n-1) + T(n-2)$$

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3.4

GroupSum5 has three conditions, in the first one it checks if the actual number is multiple of five and if the next number is one, if it's true it sums the multiple of five and skips the one (by adding two to the iterator). The other condition checks if the actual number is multiple of five and if true it sums the multiple of five and pass to the next number. And finally it has a condition that checks if by adding the actual number to the sum it will achieve the goal or by not adding it, and in both cases continue with the next number.

3.5

The following complexities are the ones that express the worst case for each algorithm.

Recursion 1

```
 Recursion-1 > Factorial
    public int factorial(int n) {
       if(n == 1 || n == 2) return n; // C1
       else return factorial(n-1)*n; // C2*T(n-1)
    }
```

T(n) = C1 + C2*T(n-1) => solved by WolphramAlpha
T(n) = C1*n + C1 => O notation, sum and multiply rules
O(n)

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

T(n) = C1 + T(n-1) + C3 => solved by WolphramAlpha
T(n) = (C1 + C2)*n + C1 => O notation, sum and multiply rules
O(n)

```
   Recursion-1 > fibonacci
    public int fibonacci(int n) {
       if(n == 0) return 0; // C1
       if(n == 1 || n == 2) return 1; // C2
       else return fibonacci(n-2) + fibonacci(n-1); // C3*T(n-1) + C3*T(n-2)
    }
```

T(n) = C1 + T(n-1) + T(n-2) => solved by "Laboratory practice No. 1: Recursion" point 4.4.1
O(2^n)

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

T(n) = C1 + T(n-1) => solved by WolphramAlpha

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

T(n) = C1*n + C1 => O notation, sum and multiply rules
O(n)

Recursion-1 > triangle

```
public int triangle(int rows) {
    if (rows == 0) return 0; // C1
    if(rows == 1) return 1; // C1
    else return triangle(rows - 1) + rows; // C2*T(n - 1) + C3
}
```

T(n) = C1 + T(n-1) => solved by WolphramAlpha
T(n) = C1*n + C1 => O notation, sum and multiply rules
O(n)

```
//  Recursion 2
```

Recursion-2 > groupSum

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

T(n) = C1 + T(n-1) + T(n-1) => solved by WolphramAlpha
T(n) = C1(2^n-1) + C1 2^(n-1) => O notation, sum and multiply rules
O(2^n)

Recursion-2 > groupSum6

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

T(n) = C1 + T(n-1) + T(n-1) => solved by WolphramAlpha
T(n) = C1(2^n-1) + C1 2^(n-1) => O notation, sum and multiply rules
O(2^n)

Recursion-2 > groupNoAdj

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

T(n) = C1 + T(n-1) + T(n-2) => solved by "Laboratory practice No. 1: Recursion" point 4.4.1
O(2^n)

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Recursion-2 > groupSum5

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

T(n) = C1 + T(n-1) + T(n-1) => solved by WolphramAlpha
T(n) = C1(2^n-1) + C1 2^(n-1)=> O notation, sum and multiply rules
O(2^n)

Recursion-2 > groupSumClump

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

T(n) = C1 + T(n-1) + T(n-1) => solved by WolphramAlpha
T(n) = C1(2^n-1) + C1 2^(n-1)=> O notation, sum and multiply rules
O(2^n)

In this algorithm, the 'm' in the "C4*T(n – m)" would be the size of the array (nums.length) minus the actual position in the array (start).

Recursion-2 > splitArray

```
public boolean splitArrayHelper(int start, int sum1, int sum2, int[] nums){
    if(start == nums.length) return sum1 == sum2; // C1
    else{
        return (splitArrayHelper(start + 1, sum1 + nums[start], sum2, nums) || // C2*T(n -1)
                splitArrayHelper(start + 1, sum1, sum2 + nums[start], nums)); // C3*T(n -1)
```

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

```
      }
   }
```

T(n) = C1 + T(n-1) + T(n-1) => solved by WolphramAlpha
T(n) = C1(2^n-1) + C1 2^(n-1)=> O notation, sum and multiply rules
O(2^n)

3.6

N: the size of the array that receives the algorithm.
M: the size of the array (nums.length) minus the actual position in the array (start).

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

4) Practice for midterms

4.2.1 A.
4.2.2 C.
4.3 A.
4.2.1 A.
4.2.2 A, C.
4.3 B.
4.4.1 C.
4.5.1 A.
4.5.2 B.
4.6 A.
4.7 E.
4.8 B.
4.6.1 sumaAux(n, i + 2)
4.6.2 sumaAux(n, i + 1)
4.7.1 comb(S, i + 1, t – S[i])
4.7.2 comb(S, i + 1, t)
4.8 C.
4.9 A.
4.10 C.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473