Laboratory practice No. 1: Recursion and Complexity

Juan David Echeverri Villada
Universidad Eafit
Medellín, Colombia
jdecheverv@eafit.edu.co

Octavio Vásquez Zapata
Universidad Eafit
Medellín, Colombia
ovasquezz@eafit.edu.co

3) Practice for final project defense presentation

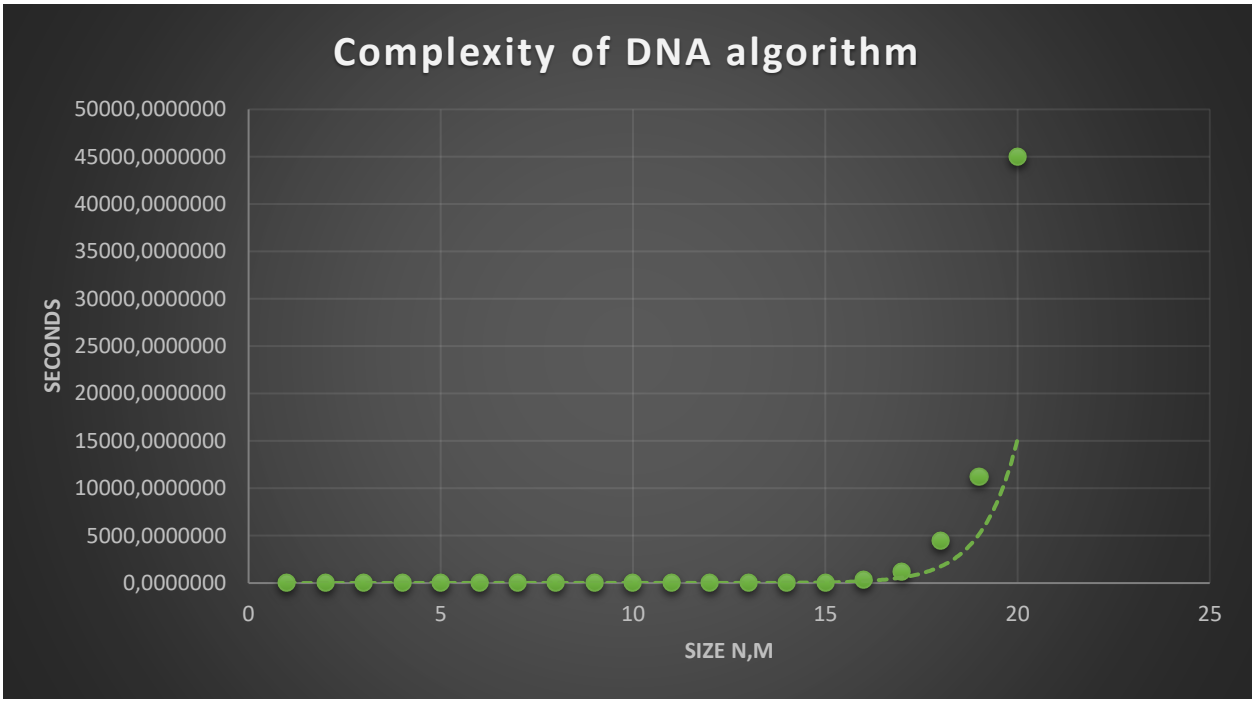
3.1 Complexity of point 1.1 (DN A)

t(n) = t((n+m) -1) + t((n+m) -1)
t(n) = 2t((n+m) -1)
t(n) = c1 * 2^ (n+m) -1
O(c1 * 2^ (n+m)-1) by O's definition
O(2^ (n+m) -1) by sum's rule
O(2^(n+m)) by sum's rule

3.2

Size n,m	Seconds	Size n,m	Seconds
1	0,0005977	11	0,6824205
2	0,000235	12	1,4377162
3	0,0005037	13	3,8343375
4	0,000293	14	12,1574442
5	0,0004519	15	72,6032069
6	0,0009028	16	355,8197943
7	0,0021651	17	1210,9174599
8	0,0112739	18	4446,732192
9	0,0341067	19	11242,8513805
10	0,1386724	20	44971,4055220

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



In the graphic, we can see that the complexity of the algorithm clearly grows exponentially, taking more and more time depending of the size of the two strings used. for taking these times, we used two random strings with same size from 1 to 19, and due to the fact that the length 20 could take easily 16 hours, we decided to make an estimation of the time that it would take. If we need to run this algorithm using two strings as big as the ones in the datasets (300.000 characters each), it would take, in the worst case, around 3805 years, which is clearly not viable at all. You could use dynamic programming to reduce the time complexity, but if you really need to run that algorithm and get the solution in a great time, you would need a super computer.

3.3 Is the complexity of the point 1.1 convenient to find the longest common subsequence between mitochondrial DNAs as the ones in the datasets?

No, definitively no, due the fact that the complexity of the algorithm 1.1(and) is O(2^(n+m)), one of the highest complexity kinds. For that reason, using that algorithm in datasets as big as the ones provides would consume a lot of memory and could take a quite long time to finish its execution. For that reason, development this algorithm using recursion and using it with big datasets is not actually very convenient

3.4 GroupSum5

GroupSum5 verify if is possible to reach a target sum with the numbers of one given array but with two conditions. Every multiple of 5 must be included and if the value next to a 5 is 1, it must be ignored.

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

3.5 Complexity of the codingBat exercises

Recursion 1

Factorial
t(n) = t(n-1) + c2
t(n) = c2*n + c1
O(c2*n + c1) by O's definition
O(c2*n) by sum's rule
O(n) by product's rule

BunnyEars
t(n) = (n-1) + c2
t(n) = c2*n + c1
O(c2*n + c1) by O's definition
O(c2*n) by sum's rule
O(n) by product's rule

Fibonacci
t(n) = t(n-1) + t(n-2)
t(n) = 2^n
O(2^n) O's definition

BunnyEars2
t t(n) = (n-1) + c2
t(n) = c2*n + c1
O(c2*n + c1) by O's definition
O(c2*n) by sum's rule
O(n) by product's rule

Triangle
t(n) = c2 + t(n-1)
t(n) = c2n + c1
O(c2*n + c1) by O's definition
O(c2*n) by sum's rule
O(n) by product's rule

Recursion 2

GroupSum6
t(n) = t(n-1) + t(n-1)
t(n) = 2t(n-1)
t(n) = c1 * 2^n-1
O(c1 * 2^n-1) by O's definition
O(2^n-1) by sum's rule
O(2^n) by sum's rule

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

GroupSumClump
t(n) = t(n-1) + t(n-1)
t(n) = 2t(n-1)
t(n) = c1 * 2^n-1
O(c1 * 2^n-1) by O's definition
O(2^n-1) by sum's rule
O(2^n) by sum's rule

GroupSum5
t(n) = t(n-1) + t(n-1)
t(n) = 2t(n-1)
t(n) = c1 * 2^n-1
O(c1 * 2^n-1) by O's definition
O(2^n-1) by sum's rule
O(2^n) by sum's rule

GroupNoAdj
t(n) = t(n-1) + t(n-2)
t(n) = 2^n
O(2^n) O's definition

SplitArray
t(n) = t(n-1) + t(n-1)
t(n) = 2t(n-1)
t(n) = c1 * 2^n-1
O(c1 * 2^n-1) by O's definition
O(2^n-1) by sum's rule
O(2^n) by sum's rule

3.6 Variables 'n' and 'm'

When calculating an algorithm complexity, the variable 'n' means the size of the dataset that is going to be used. When the complexity of the algorithm depends on two different parts of the datasets, the variable 'n' is usually used to represent the first part and the 'm' the second one

4) Practice for midterms

4.1)
4.1.1 a
4.1.2 c
4.1.3 a

4.2)
4.2.1 b
4.2.2 a- b-c

4.3)
b

4.4)
lucas(n-1) + lucas(n-2)

4.4.1
c

4.5)
4.5.1 a
4.5.2 b

4.6)
a

4.7)
e

4.8)
03 ---------- if(t<0) return false;
04 ---------- if(t==0) return true;
08 ---------- return f1 || f2 || f3;

4.8.1
b

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

(after optional point 4.8, some of the points start getting repeated. we decided to preserve the continual numeration)

4.9)

4.9.1
10 ---------- sumaaux(n.substring(i+2, n.length()), i+2)
4.9.2
12 ---------- sumaaux(n.substring(i+1, n.length()), i+1)

4.10)

4.10.1
9 ---------- comb(s, i+1, t-s[i])

4.10.2
10 ---------- comb(s, i+1, t)

4.11)

c (22)

4.12)

b

4.13)

lucas(n-1) + lucas(n-2)

4.13.1

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