Algorithmics	Student information	Date	Number of session
	UO:		
	Surname:		
	Name:		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. [Divide and Conquer by subtraction]

In Subtraction1, a=1,b=1,k=0 so its complexity is O(n). It stops the execution because of a stack overflow when n=8192. I cannot know if the times matches the theortical result as we cannot get reliable times because there is a stack overflow before we get them.

In Subtraction2, a=1,b=1,k=1, so its complexity is O(n2). It stops its execution because of a stack overflow when n is greater than 132768. With the few times that we see before the stack overflow, can see that they match the theoretical result as when we increase the size of the problem by 2, the increase by 2^2.

In Substraction3, a=2,b=1,k=0 so its complexity is $O(2^n)$. To calculate how many years it would take to execute n=80:

$n_1=30 \rightarrow t_1=1{,}872$ s

$n_2=80 \rightarrow t_2=?$

$t_2=(2^{n2}/2^{n1})*t_1=2^{(n2-n1)}*t_1=2^{50}*1{,}872 = 2.10768463*10^{15}$ s = 66834257,94 years

Substraction 4:

n	T(s)
100	LoR
200	LoR
400	0,11
800	0,864
1600	6,857
3200	54,982
6400	OoT

It is what we expect as when we. Increase the size by two, the time grows by 2^3.

Algorithmics	Student information	Date	Number of session
	UO:		
	Surname:		
	Name:		

Substraction5:

n	T(s)
30	0,445
32	1,324
34	3,971
36	11,921
38	35,64
40	OoT

It is what we expect as when we. Increase the size by two, the time grows by 3, which is 3^(2/2).

n_1=38 → t_1=35,64s

n_2=80 → t_2=?

$t_2=(3^{n2/2}/3^{n1/2})*t_1=3^{(n2-n1)/2}*t_1=3^{21}*35,64$ = 3.72806988$*^{11}$ s = 11821,63 years

Activity 2. [Divide and conquer by division]

In Division1, a=1; b=3; k=1 so its complexity is O(n). We can see that it matches the expected result, as the times that we can measure when we increase n by 2, they grow by 2.

In Division2, a=2; b=2; k=1 so its complexity is O(nlogn). We can see that it matches the expected result, as the times when we increase n by 2, they grow a bit faster than 2 .

In Division3, a=2; b=2; k=0 so its complexity is O(n). We can see that it matches the expected result, as the times when we increase n by 2, they grow by 2.

In Division4, a=1;b=2;k=2 so its complexity is O(n^2)

n	T(s)
1000	LoR
2000	LoR
4000	0,171
8000	0,668
16000	2,584
32000	10,320
64000	42,586
128000	OoT

As we can see, the results match the expected valuesa as when we increase size by 2, the times grow by 2^2.

In Division5, a=4,b=2,k=1 so its complexity is O(n^2)

n	T(s)
1000	0,051
2000	0,205
4000	0,783
8000	3,221
16000	12,895
32000	50,971
64000	OoT
128000	OoT

As we can see, the results match the expected valuesa as when we increase size by 2, the times grow by 2^2.

Algorithmics	Student information	Date	Number of session
	UO:		
	Surname:		
	Name:		

Activity 4. [Two basic examples]

n	Tsum1(s)	Tsum2(s)	Tsum3(s)
1000	0,0000111	0,0000255	0,0000586
2000	0,0000205	0,0000509	0,0001116
4000	0,0000396	0,0001005	0,0002277
8000	0,0000753	0,0001962	0,0004438
16000	0,0001423	Stack Overflow	0,0009258
32000	0,0002829	Stack Overflow	0,0018058
64000	0,000597	Stack Overflow	0,003565

All methods are linear, but this complexity is acheived by different forms. In sum1 using a for loop, in sum2 using divide and conquer by substraction, and in sum3 by division. We can see that using substraction causes overflow due to the number of recursive calls. And the more efficient algorithm is sum1 taking as a reference the times that I have obtained.

n	Tfib1(s)	Tfib2(s)	Tfib3(s)	Tfib4(s)
10	0,000000149	0,000000201	0,000000319	0,0000048
20	0,000000246	0,000000360	0,000000538	0,000534
30	0,000000346	0,000000529	0,000000748	0,0657
40	0,000000445	0,000000698	0,000001032	8,356
50	0,000000541	0,000000836	0,000001242	OoT

Methods fib1, fib2 and fib3 have a linear complexity, fib1 and fib2 using a for loop and fib3 using a recursive solution. Fib4 has an exponential complexity O(1.6^n) so we know that this method is worse than the other three. Observing the times, we can stay that the more efficient algorithm is fib1.