


Algorithmics	Student information	Date	Number of session
	UO:300895	13-325	7
	Surname: Franco Martinez	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Ivan		



Activity 1. [Divide and Conquer by subtraction]

For what value of n do the Subtraction1 and Subtraction2 classes stop giving times (we abort the algorithm because it exceeds 1 minute)? Why does that happen?

For subtraction1 and subtraction2 we stop at $n = 8192$, it happens because the programs overflow when the problem size grows to several thousands.

Subtraction4		Subtraction5	
n	t ordered	n	t ordered
100	1,45	30	386
200	10,54	32	1128
400	82,28	34	3389
800	644	36	10363
1600	5152	38	30625
3200	40553	40	OoT
6400	OoT		

How many years would it take to complete the Subtraction3 execution for $n=80$? Reason the answer

As it have a complexity $O(2^n)$: Total time = $280 \times 10^{-6} = 1.2089 \times 10^{18}$ seconds

Total time in years = $(1.2089 \times 10^{18}) / 31557600 \approx 3.83 \times 10^{10}$ years

How many years would it take to complete the Subtraction5 execution for $n=80$? Reason the answer.

As it have a complexity $O(3^{n/2})$:

$340 \approx 1.21 \times 10^{19}$

Total time = $1.21 \times 10^{19} \times 10^{-6}$ seconds = 1.21×10^{13} seconds

Total time in years = $(1.21 \times 10^{13}) / 31557600 \approx 383000$ years

Algorithmics	Student information	Date	Number of session
	UO:300895	13-325	7
	Surname: Franco Martinez		
	Name: Ivan		

Activity 2. [Divide and conquer by division]

After analyzing the complexity of the three previous classes, you are not asked to make the timetables, but to reason whether the times match the theoretical time complexity of each algorithm. Implement a Division4.java class with a complexity $O(n^2)$ (with abk) and then fill in a table showing the time (in milliseconds) for $n=1000, 2000, 4000, 8000, \dots$ (up to OoT).

In Division1 as we have $a < b^k$ and $a=1; b=3; k=1$ according to D and C by division we have a complexity of $O(n)$

In Division2 as we have $a = b^k$ and $a=2; b=2; k=1$ according to D and C by division we have a complexity of $O(n \cdot \log n)$

In Division3 as we have $a > b^k$ and $a=2; b=2; k=0$ according to D and C by division we have a complexity of $O(n^{\log_2 2}) = O(\log n)$

We can see how the complexities match with the times obtained

Division5	
n	t ordered
1000	26
2000	98
4000	385
8000	1541
16000	6169
32000	24821
64000	OoT

Division4	
n	t ordered
1000	0,4
2000	15,3
4000	60,3
8000	238,9
16000	963,4
32000	3826,3
64000	17004
128000	OoT

Activity 3. [Two basic examples]

After analyzing the complexity of the various algorithms within the two classes, executing them and after putting the times obtained in a table, compare the efficiency of each algorithm.

Algorithmics	Student information	Date	Number of session
	UO:300895	13-325	7
	Surname: Franco Martinez		
	Name: Ivan		

VecorSum2					
n	1	n	2	n	3
1536	0,0064	384	0,0068	192	0,0062
3072	0,0121	768	0,0133	384	0,0124
6144	0,0238	1536	0,0266	768	0,0243
12288	0,0644	3072	0,0519	1536	0,049
24576	0,5152	6144	0,1058	3072	0,0975
49152	4,0553	12287	0,2038	6144	0,198
98304	3,726	24579	0,4138	12288	0,3939

Fibonacci1				
n	1	2	3	4
10	0,000083	0,000115	0,000186	0,00244
11	0,000089	0,000115	0,000218	0,0038
12	0,000091	0,000121	0,000231	0,00612
13	0,000092	0,000127	0,000246	0,01004
14	0,000096	0,000115	0,000261	0,01676
15	0,000101	0,000136	0,00028	0,02677
16	0,000105	0,000141	0,000289	0,04188
17	0,000108	0,000156	0,000305	0,06914
18	0,000112	0,000161	0,000319	0,1187
19	0,000116	0,000163	0,000343	0,1811
20	0,00012	0,000175	0,000353	0,29412
21	0,000122	0,000179	0,000366	0,47128
22	0,00013	0,000182	0,000385	0,76732
23	0,000133	0,000191	0,000406	1,3874
24	0,000136	0,000204	0,000411	2,008
25	0,000144	0,000209	0,000434	3,567

In the VectorSum we have three different types of implementation.

In Sum1 we iterate through a for loop. $O(n)$

In Sum2 we use a divide and conquer strategy by subtraction $a=1$ so $O(n^{k+1}) = O(n)$

In Sum3 we use a divide and conquer strategy by division $a > b^k$ so $O(n^{\log a}) = O(n)$

The different types of implementations makes the times fluctuate although they have the same complexity($O(n)$)

In the Fibonacci it happens something similar as in VectorSum, we have difference types of implementations the first three with the same complexity $O(n)$ but with difference times.

Algorithmics	Student information	Date	Number of session
	UO:300895	13-325	7
	Surname: Franco Martinez		
	Name: Ivan		

In fib1 it use a simple for loop, in fib2 it use a loop but with a vector using dynamic programming, In fib3 it use a use a divide and conquer strategy by subtraction($a=1$)
In fib4 the time complexity is $n(1.6^n)$ and it use a divide and conquer strategy by division ($a > b^k$)

Activity 4. [Petanque championship organization]

n	t Calendar
2	0,002
4	0,006
16	0,026
32	0,05
64	0,129
128	0,369
256	1,54
512	5,98
1024	29,429
2048	OoT

This algorithm which use a D&C strategy to build a calendar has a time complexity of $O(n \log n)$ something which meet the times obtained in the measurements