Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersán		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Divide and conquer by subtraction

The problem with Subtractions 1 and 2 is that after a certain size is evaluated, a StackOverFlow error is launched, as explained in the script.

For subtraction3 (after running the code and analyzing the measurements) we will assume n1=32, t1=6498 ms, n2=80. The complexity of the algorithm is $O(2^n)$ with a=2, b=1 and k=0. This means that $t2=(2^{80}/2^{32})*6498$. Whatever result we get we shall convert it from milliseconds to years, and that would be the answer to the problem.

Subtraction 4 ($O(n^3)$):

n	t (ms)
100	1
200	4
400	1
800	3
1600	26
3200	193
6400	1411
12800	10842

Subtraction 5 ($O(3^{n/2})$):

n	t (ms)
30	19
32	62
34	63
36	542
38	544
40	4657
42	4651
44	42442
46	43352

For subtraction5 (after running the code and analyzing the measurements) we will assume n1=46, t1=43352 ms, n2=80. The complexity of the algorithm is $O(3^{n/2})$ with a=3, b=2 and k=0. This means that $t2=(3^{40}/2^{23})*43352$. Whatever result we get we shall convert it from milliseconds to years, and that would be the answer to the problem.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersán		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 2. Divide and conquer by division

Division 4 $O(n^2)$ -> $a<b^k$

n	t (ms)
1000	0
2000	0
4000	0
8000	0
16000	0
32000	20
64000	81
128000	283
256000	1107
512000	4388
1024000	18051

We can see that in the first few sizes the problem is solved too quick to notice much, but from n=32000 onwards we can clearly see the $O(n^2)$ complexity.

Division 5 $O(n^2)$ -> $a>b^k$

n	t (ms)
1000	0
2000	15
4000	34
8000	51
16000	547
32000	869
64000	8447
128000	14275

We can clearly see that these times do not match the $O(n^2)$ complexity, even though the code is built correctly for it. If the times were right, we should notice that the time value should approximately be around 4 times the previous one, since we are duplicating the size of the problem. So it t at n=16000 is worth 547ms, t at n=32000 should be around 2188 ms, and so on.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersán		

Activity 3. Vectors and Fibonacci

VECTOR SUM:

Option -> 1 nTimes -> 100000

n	t (ms)
1000	32
2000	51
4000	99
8000	196
16000	393
32000	785
64000	1563
128000	3125
256000	6233
512000	12374

Option -> 2 nTimes -> 100000

n	t (ms)
1000	155
2000	404
4000	714
8000	1210
16000	3524

After n=16000, we get a StackOverFlow error, so the execution cannot continue. This proves that option 1 is better than option 2.

Option -> 3 nTimes -> 100000

n	t (ms)
1000	248
2000	392
4000	1035
8000	1640
16000	4144
32000	6495
64000	16489
128000	25203

The linear complexity here is not very consistent, as we can see from the times measured above. So overall, we can say that the best option for summing up the vector is the first one.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersán		

FIBBONACI:

Option -> 1 nTimes -> 100000000

n	t(ms)
10	245
11	259
12	315
13	332
14	382
15	414
...	...
55	836
56	899
57	762
58	792
59	816

Here we can see that the linear complexity is pretty accurate.

Option -> 2 nTimes -> 100000000

n	t(ms)
10	415
11	478
12	550
13	640
14	622
15	703
...	...
55	2812
56	2947
57	3087
58	2890
59	2954

Here we are adding the results we get int each iteration in a vector and the final result will be stored in the last position. The complexity is still linear, but in this case we can see that for the bigger sizes, it take much more time than the previous option.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersán		

Option -> 3 nTimes -> 100000000

n	t(ms)
10	715
11	727
12	830
13	870
14	953
15	1026
...	...
55	3824
56	4092
57	4093
58	4208
59	4135

This is the first recursive method. It is done by passing the previous numbers as a parameter. As we can see, it is much slower than the previous one.

Option -> 4 nTimes -> 1000000

n	t(ms)
10	138
11	217
12	362
13	599
14	967
15	1610
16	2698
17	4519
18	6740
19	10714
20	17542
21	27974
22	46175

This option does not follow a linear complexity, instead it follows $O(1.6^n)$. It recursively calls the sequence of the previous number and adds it to the result of the previous number if that (also recursively). We can see that this is slowest option out of the ones that we are given.