


Algorithmics	Student information	Date	Number of session
	UO: 293860	27/02/2024	3
	Surname: López Álvarez	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Juan		



Activity 1. Divide and Conquer by subtraction

Subtraction1 and Subtraction2 get a stack overflow error for $n = 16384$. This happens because every time we make a recursive call to the method the value of the counter is stored in the stack so that when the execution of the recursive call ends, the method can continue, the problem is that if we push too many values to the stack there is a moment where there is no more space and the program crashes giving us the overflow error.

The complexity of Subtraction3 is $O(n^2)$, so if we take the time t_1 corresponding to any size n_1 we can compute the time it would take to execute the algorithm for $n=80$. We take the values $n_1 = 25$, $t_1 = 1845\text{ms}$, $n_2 = 80$. $T_2 = (n_2/n_1)^n * t_1 = (80/25)^{80} * 1845 = 1.5e^{33}$.

Subtraction4.java			Subtraction5.java		
n	Time (ms)		n	Time (ms)	
100	3		30	535	
200	31	24	32	1520	1605
400	170	248	34	4782	4560
800	1216	1360	36	15635	14346
1600	9777	9728	38	43033	46905
3200	Oot	78216	40	Oot	129099

Next to the table I computed the estimated time to check if the algorithm matches the complexity expected. To know how many years it would take to run Subtraction5 with $n=80$ we do the following computation: $n_1 = 34$, $t_1 = 4782$, $n_2 = 80 \Rightarrow t_2 = (3^{(n_2/2)} / 3^{(n_1/2)}) * t_1 = 14275.5$ years more or less.

Activity 2. Divide and conquer by division

The algorithms match the complexity expected, however times are very low so we can only see a few relevant time measurements.

Algorithmics	Student information	Date	Number of session
	UO: 293860	27/02/2024	3
	Surname: López Álvarez		
	Name: Juan		

Division4.java			Division5.java		
n	Time (ms)		n	Time (ms)	
1000	10		1000	32	
2000	39	40	2000	129	128
4000	151	156	4000	508	516
8000	590	604	8000	2106	2032
16000	1340	2360	16000	8229	8424
32000	9433	5360	32000	35148	32916
64000	37653	37732	64000	Oot	140592
128000	Oot	150612			

Again I computed the expected time next to the ones obtained in the real execution.

Activity 3. VectorSum & Fibonacci

sum1		sum2		sum3	
n	Time (ms)	n	Time (ms)	n	Time (ms)
3	1	3	4	3	3
6	2	6	6	6	5
12	3	12	11	12	10
24	6	24	15	24	21
48	9	48	22	48	42
96	18	96	44	96	84
192	34	192	97	192	167
384	67	384	199	384	338
768	130	768	410	768	1141
1536	252	1536	835	1536	1353
3072	504	3072	1719	3072	2782
6144	1013	6144	3524	6144	5908
12288	2009	12288	-	12288	10905
24576	4001	24576	-	24576	23913
49152	7995	49152	-	49152	45095
98304	15986	98304	-	98304	Oot

The execution of the sum2 algorithm causes a stack overflow when n is greater than 6144, this happens for the same reason as in the first subtraction algorithms. In terms of time efficiency we can clearly see that the best algorithm is the first one, which is in fact the simplest. This difference can be caused due to the time spent calling the recursive function rather than simply iterating over the vector.

Activity 4. Mergesort

Algorithmics	Student information	Date	Number of session
	UO: 293860	27/02/2024	3
	Surname: López Álvarez		
	Name: Juan		

Mergesort			
n	t ordered	t reverse	t random
31250	17	19	20
62500	33	34	44
125000	75	75	94
250000	139	165	179
500000	283	287	366
1000000	612	631	771
2000000	1200	1277	1559
4000000	2540	2634	3220
8000000	5336	5443	6624
16000000	10653	11103	13770

n	t Mergesort (t1)	t Quicksort (t2)	t1/t2
250000	168	117	1,4359
500000	376	248	1,5161
1000000	878	532	1,6504
2000000	1627	1167	1,3942
4000000	3225	2617	1,2323
8000000	6627	6350	1,0436
16000000	13596	16962	0,8016

As we can see the mergesort algorithm is faster for all the sizes but the last one, the reason for this improvement might be the use of external memory, because we are using auxiliary vectors to combine the solutions of the recursive calls, while in the quicksort algorithm every operation (swaps, comparisons...) is done in the input vector.