PRACTICA 3

Modelos recursivos básicos

En este apartado de la práctica nos encargamos de analizar distintos algoritmos recursivos, unos que reducían el tamaño del problema restando valores a este (algoritmos recursivos por sustracción), y otros que lo reducían dividiendo el tamaño del problema (algoritmos recursivos por división).

Para poder entender la complejidad de cada uno hay que saber como analizar un algoritmo Divide y Vencerás:

- En el caso de recursividad por sustracción:
  - Tenemos cuatro parámetros que necesitamos conocer:
    - n: tamaño del problema.
    - a: es el número de llamadas recursivas que se realizan en el algoritmo.
    - b: es el valor en el que se reduce el tamaño del problema (se resta).
    - k: es el exponente que tendría la complejidad si no hubiese llamadas recursivas.
  - En función de los parámetros anteriores, la complejidad es la siguiente:
    - Si a=1: la complejidad es:

      $O(n^{k+1})$
    - Si a>1: la complejidad es:
      $O(a^{n/b})$
- En el caso de recursividad por división:
  - Tenemos cuatro parámetros que necesitamos conocer:
    - n: tamaño del problema.
    - a: es el número de llamadas recursivas que se realizan en el algoritmo.
    - b: es el valor en el que se reduce el tamaño del problema (se divide).
    - k: es el exponente que tendría la complejidad si no hubiese llamadas recursivas.
  - En función de los parámetros anteriores, la complejidad es la siguiente:
    - Si $a<b^k$: la complejidad es:

      $O(n^k)$
    - Si $a=b^k$: la complejidad es:
      $O(n^k * \log(n))$
    - Si $a>b^k$: la complejidad es:
      $O(n^{\log_b(a)})$

Los dos primeros algoritmos analizados con un esquema Divide y Vencerás por sustracción tenían un a=1 (solo una llamada recursiva), mientras que el tercero tenía un a>1 (más de una llamada recursiva). Además, sabemos que los algoritmos recursivos por sustracción con un a=1 conllevan un gran gasto de la pila, que limita la potencia de estos en gran medida (suele haber mejores soluciones iterativas), mientras que los algoritmos recursivos por sustracción con un a>1 suelen ser muy lentos, ya que tienen una complejidad exponencial.
Posteriormente se analizaron tres algoritmos recursivos por división, de forma que se comprobaron los tres tipos de complejidades que pueden tener estos (las mostradas

anteriormente).
Después, también se analizaron las complejidades y los tiempos de ejecución de tres formas diferentes de sumar los elementos de un vector y también de cuatro formas diferentes de calcular el fibonacci de un número.
Por último, se crearon dos nuevos algoritmos (uno recursivo por sustracción y otro por división) para que tuviesen la complejidad que se nos había proporcionado.

Los resultados de los análisis fueron los siguientes:

- Sustracción 1:
  - a=1
  - b=1
  - k=0
  - Complejidad: O(n)
- Sustracción 2:
  - a=1
  - b=1
  - k=1
  - Complejidad: $O(n^2)$
- Sustracción 3:
  - a=2
  - b=1
  - k=0
  - Complejidad: $O(2^n)$
- División 1:
  - a=1
  - b=3
  - k=1
  - Complejidad: O(n)
- División 2:
  - a=2
  - b=2
  - k=1
  - Complejidad: O(n * log n)
- División 3:
  - a=2
  - b=2
  - k=0
  - Complejidad: O(n)
- Suma vector 1:
  - Método iterativo con complejidad O(n).
- Suma vector 2:
  - Método recursivo con complejidad O(n).
    - Recursividad por sustracción.
      - a=1
      - b=1
      - k=0
- Suma vector 3:
  - Método recursivo con complejidad O(n).
    - Recursividad por división.
      - a=2

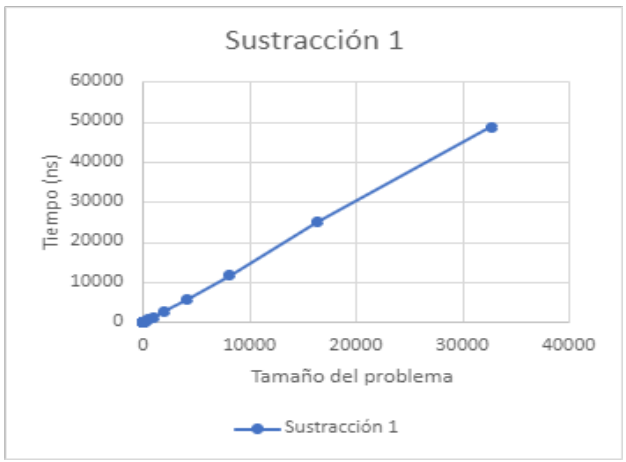
        - b=2
        - k=0
- Fibonacci 1:
    - Método iterativo con complejidad O(n).
- Fibonacci 2:
    - Método iterativo con complejidad O(n). En este caso se usa un vector para ir almacenando los fibonacci de cada número según se van calculando.
- Fibonacci 3:
    - Método recursivo con complejidad O(n).
        - Recursividad por sustracción.
            - a=1
            - b=1
            - k=0
- Fibonacci 4:
    - Método recursivo con complejidad exponencial: $O(1.6^n)$
    - No se puede calcular directamente la complejidad, sino que se tiene que hacer una aproximación de esta calculando la complejidad al alza y a la baja. Esto se debe a que en el algoritmo se hacen dos llamadas recursivas que tienen distinta reducción del problema (distinto b).

Algoritmos creados:

- Sustracción 4:
  Se pedía un método recursivo por sustracción que tuviese una complejidad $O(3^{n/2})$. Para ello, los parámetros tenían que ser los siguientes:
    - a=3
    - b=2
    - k=0 (aunque no es relevante porque a>1)
- División 4:
  Se pedía un método recursivo por división que tuviese una complejidad $O(n^2)$ y que tuviese cuatro llamadas recursivas. Para ello, los parámetros tenían que ser los siguientes:
    - a=4
    - b=3
    - k=2

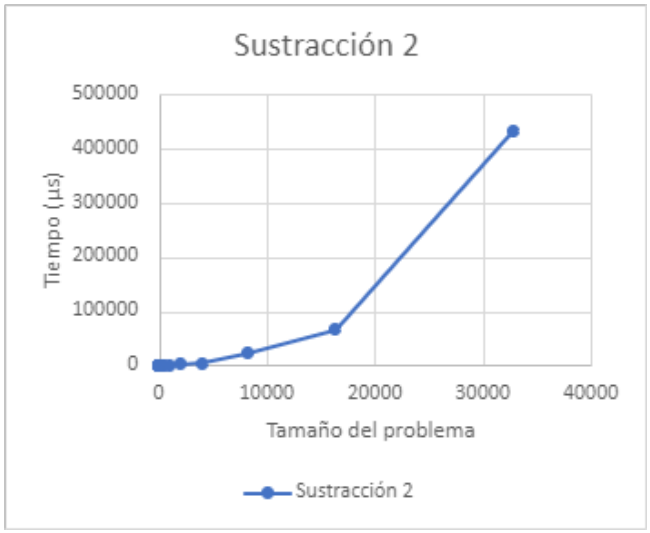
Tiempos medidos y gráficas:

Tamaño del problema	Sustracción 1 (ns)
1	4
2	4
4	3
8	4
16	10
32	19
64	50
128	112
256	269
512	563
1024	1074
2048	2734

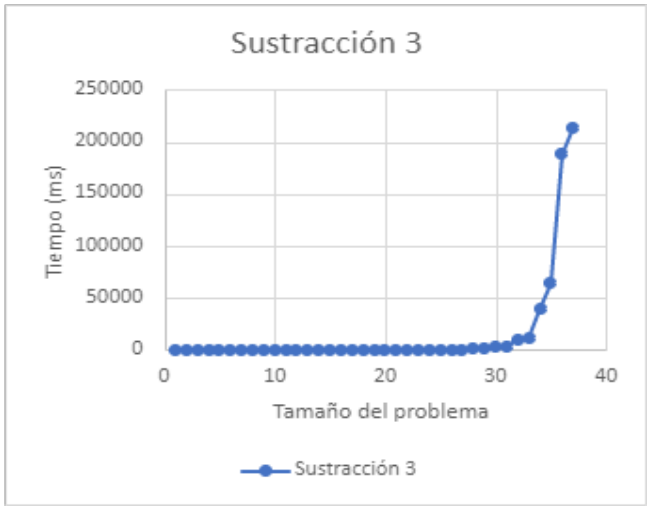


4096	5437
8192	11618
16384	24826
32768	48715
65536	StackOverflow

Tamaño del problema	Sustracción 2 (µs)
1	0
2	0
4	0
8	1
16	0
32	2
64	1
128	5
256	20
512	75
1024	268
2048	1006
4096	4137
8192	22237
16384	65308
32768	431222
65536	StackOverflow



Tamaño del problema	Sustracción 3 (ms)
25	41
26	158
27	172
28	585
29	678
30	2411
31	2663
32	9816
33	10520
34	38856
35	64131
36	188549
37	214199

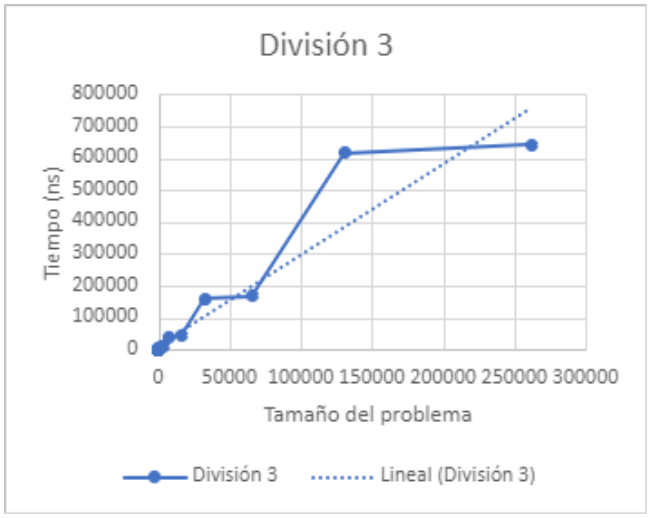


Tamaño del problema	División 1 (ns)
64	51
128	101
256	176
512	334
1024	632
2048	1351
4096	2360

8192	5644
16384	13970
32768	19656
65536	36075
131072	66376
262144	124766
524288	245995
1048576	418007
2097152	780436
4194304	1548022

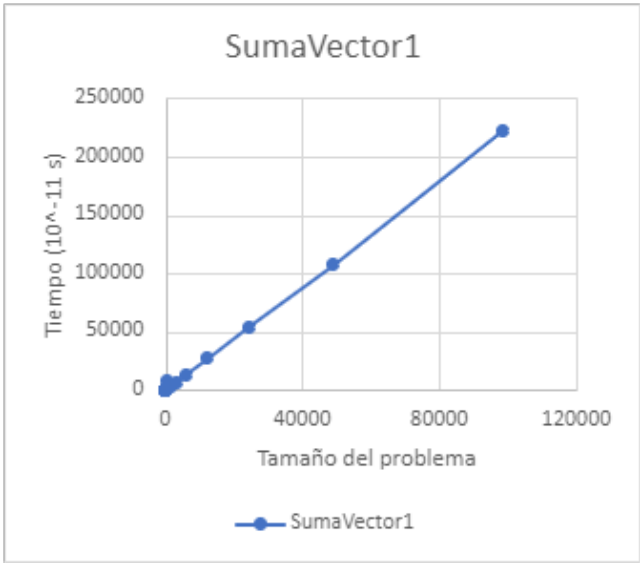
Tamaño del problema	División 2 (µs)
512	4
1024	6
2048	18
4096	27
8192	72
16384	111
32768	300
65536	481
131072	1284
262144	2035
524288	5350
1048576	8739
2097152	22530
4194304	46013
8388608	218248

Tamaño del problema	División 3 (ns)
1	3
2	8
4	6
8	35
16	37
32	148
64	160
128	594
256	620
512	2442
1024	2673
2048	10166
4096	11131
8192	41143
16384	43849
32768	158152

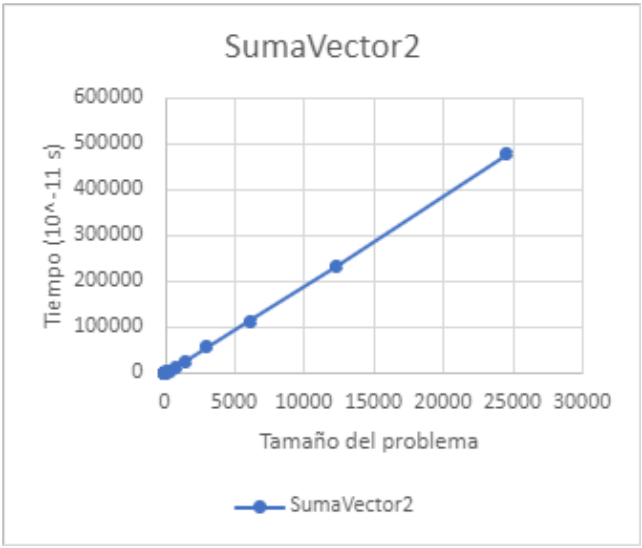


65536	167195
131072	617648
262144	641640

Tamaño del problema	SumaVector1 (10^-11 s)
3	16
6	26
12	25
24	39
48	73
96	150
192	365
384	736
768	7560
1536	3209
3072	6540
6144	13202
12288	27074
24576	53975
49152	107083
98304	221955

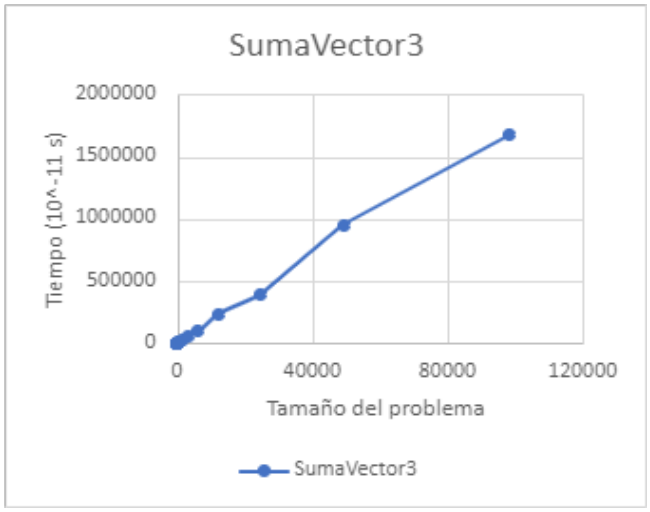


Tamaño del problema	SumaVector2 (10^-11 s)
3	21
6	44
12	83
24	162
48	337
96	775
192	1608
384	3382
768	9988
1536	24847
3072	55658
6144	113307
12288	231315
24576	478289
49152	StackOverflow

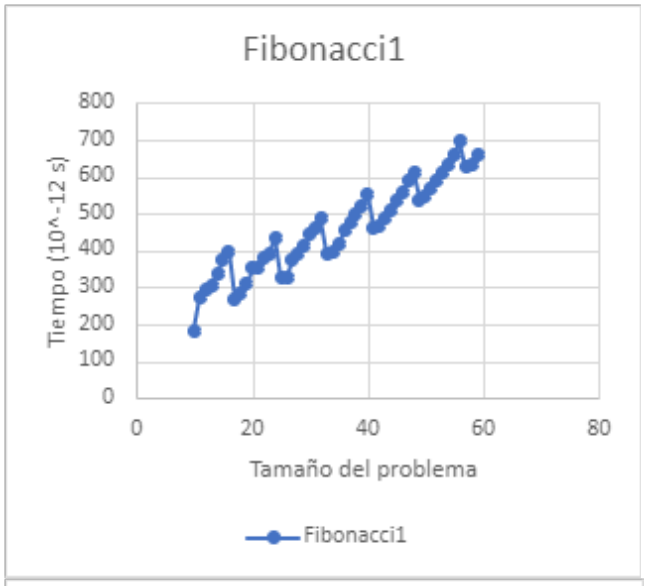


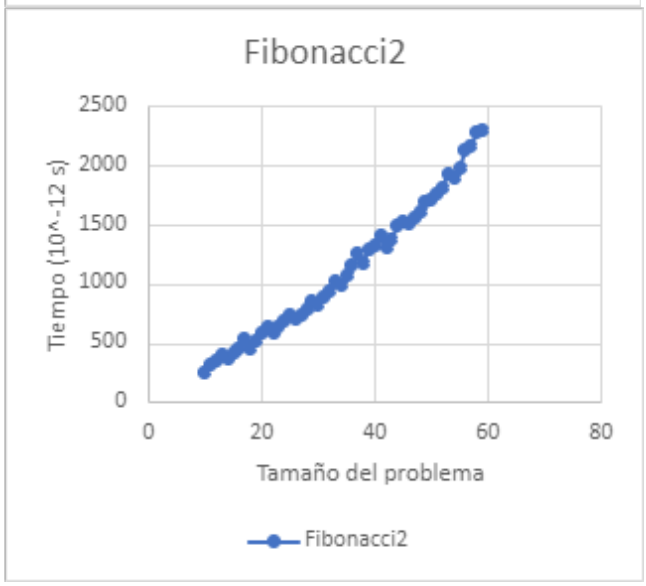
Tamaño del problema	SumaVector3 (10^-11 s)
3	37
6	85
12	200
24	364
48	899
96	1538
192	3676

384	6212
768	14888
1536	24950
3072	59814
6144	97830
12288	237685
24576	389849
49152	950288
98304	1677175

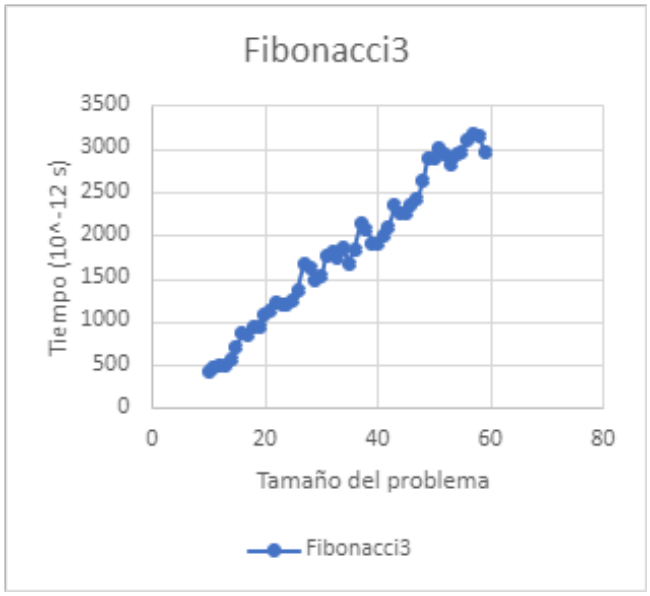


Tamaño del problema	Fibonacci 1 (10^-12 s)	Fibonacci 2 (10^-12 s)	Fibonacci 3 (10^-12 s)
10	183	242	420
11	273	314	462
12	293	356	486
13	307	396	485
14	339	360	561
15	373	409	697
16	398	455	869
17	266	530	850
18	284	451	935
19	308	517	947
20	352	581	1082
21	356	640	1134
22	380	583	1217
23	391	641	1187
24	436	693	1197
25	326	737	1242
26	329	703	1368
27	373	734	1653
28	390	792	1616
29	412	859	1487
30	445	820	1515
31	459	889	1770
32	488	942	1805
33	393	1023	1738
34	395	980	1848
35	416	1079	1671
36	455	1148	1830
37	477	1253	2125
38	498	1171	2072
39	520	1297	1908
40	550	1317	1902
41	461	1401	1999

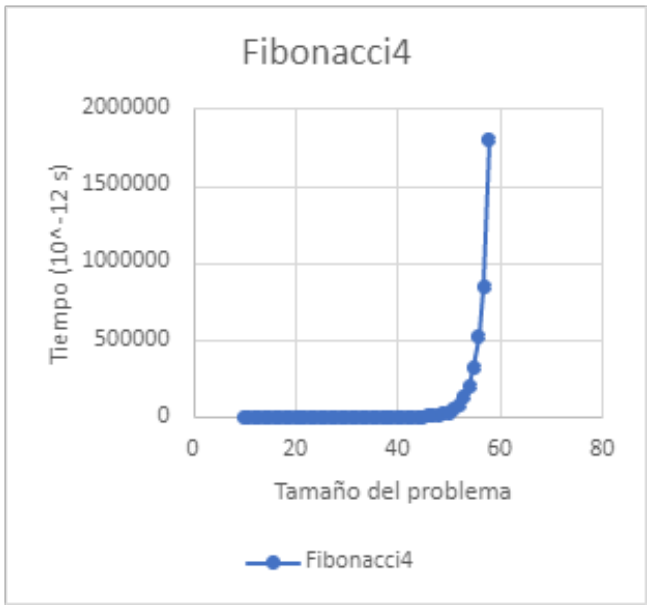




42	465	1313	2099
43	489	1377	2335
44	510	1489	2251
45	536	1519	2243
46	557	1502	2348
47	589	1562	2413
48	612	1610	2639
49	539	1694	2887
50	545	1703	2891
51	568	1757	2994
52	589	1814	2930
53	611	1932	2809
54	633	1899	2939
55	659	1973	2959
56	695	2124	3092
57	626	2171	3174
58	635	2280	3148
59	658	2297	2966

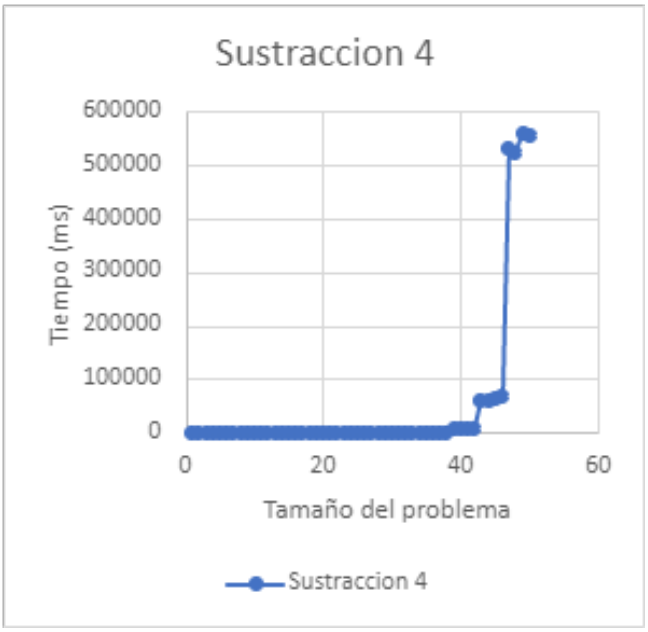


Tamaño del problema	Fibonacci 4 (10^-12 s)
35	22
36	37
37	61
38	100
39	160
40	258
41	419
42	676
43	1105
44	1782
45	2902
46	4661
47	7474
48	11915
49	19102
50	30635
51	53921
52	83003
53	124960
54	198801
55	319282
56	514667
57	840858
58	1801998



Tamaño del problema	Sustraccion 4 (ms)
30	9
31	81
32	79
33	86

34	86
35	727
36	738
37	788
38	801
39	6636
40	6807
41	7431
42	7523
43	61205
44	59419
45	62674
46	66150
47	530459
48	524081
49	558039
50	556998



Tamaño del problema	División 4 (ms)
4096	2
8192	3
16384	11
32768	41
65536	154
131072	556
262144	2216
524288	8571
1048576	34340
2097152	184777
4194304	640104

El skyline de la ciudad

En esta parte, tuvimos que desarrollar el código para poder obtener el contorno que forman los edificios de una ciudad. Para solucionar este problema, tuvimos que desarrollar dos algoritmos, uno de ellos iterativo usando la fuerza bruta, mientras que el otro es recursivo siguiendo un esquema divide y vencerás.
La complejidad del algoritmo de fuerza bruta es $O(n^2)$, ya que hay un bucle y en su interior hay dos operaciones O(n), mientras que fuera del bucle hay dos operaciones O(n), por lo que la complejidad sería $O(2n^2+2n)$, que se reduce a $O(n^2)$.

Por otra parte, el algoritmo recursivo utiliza un esquema divide y vencerás por división, en el que se realizan dos llamadas recursivas y se divide el tamaño del problema entre dos

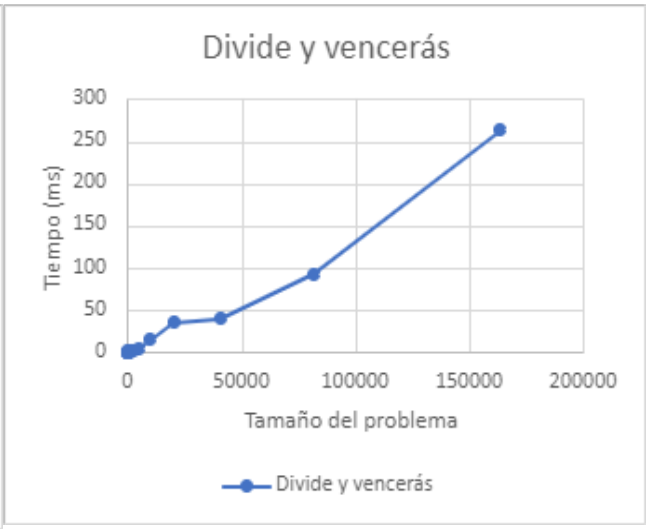
en cada llamada. Si analizamos esto siguiendo el esquema divide y vencerás explicado en la anterior práctica obtenemos el siguiente valor de los parámetros:

- a: 2
- b: 2
- k: 1

En este caso, como $a=b^k$, la complejidad sería O(n * log n). Además, antes de realizar la llamada recursiva, se ordena la colección, y como sabemos los algoritmos de ordenación tienen una complejidad O(n * log n) como mínimo (que es el caso actual). También hay una operación O(n), llamada calculateKeyPoints(), pero en este caso, al ser la complejidad más baja que el resto se descarta al reducirla. Entonces, la complejidad del algoritmo sería O(n * log n).

Los datos empíricos de mediciones de tiempo son los siguientes:

Tamaño	Fuerza bruta (ms)	Divide y vencerás (ms)
10	2	1
20	0	0
40	0	0
80	1	0
160	5	0
320	16	0
640	26	0
1280	75	1
2560	267	2
5120	1046	5
10240	4273	15
20480	18279	35
40960	78371	40
81920	607496	93
163840	2586909	263



Como se puede ver según los datos obtenidos se cumplen las complejidades explicadas anteriormente, por lo que podemos concluir que el mejor algoritmo de resolución del problema es el que utiliza un esquema divide y vencerás, ya que tiene menor complejidad que el algoritmo iterativo de fuerza bruta y por ello consume mucho menos tiempo en

resolver el problema.

*Nota: sabemos que se cumplen las complejidades de la siguiente forma:

Fuerza bruta: la complejidad era $O(n^2)$ y la n se multiplica por dos, entonces los tiempos se tienen que multiplicar por 2^2 y por una constante, lo cual se cumple.

Divide y vencerás: la complejidad era O(n * log n) y la n se multiplica por dos, entonces los tiempos se tienen que multiplicar aproximadamente por 2 *log 2 y por una constante, lo cual se cumple.