UT2-PD3

5.4

caso 1

a) $150 \cdot (10000) \cdot \log_2(10000) = 1500000 \cdot 13,29 = 19935000$

caso 2: $10000^2 = 100000000$ → Es más favorable ya que es casi la mitad de su valor

caso 1

b) $150 \cdot (100) \cdot \log_2(100) = 15000 \cdot 6,64 = 99600$

caso 2: $(100)^2 = 10000$ → Es más favorable ya que es un valor menor

c) caso 1: $150 \cdot (1000) \cdot \log_2(1000) = 150000 \cdot 9,97 = 1495500$

caso 2: $(1000)^2 = 1000000$ → Es el caso más favorable.

d) No, va a llegar a un punto donde la entrada va a ser tan grande que va a convenir utilizar el primer código.

5.5

a) Línea 14: utilizando el test inicio < fin.
Recorre todos los elementos menos el último

b) Línea 16: haciendo la asignación medio = inicio + fin/2
Iría cambiando los valores de medio, quizás sean iguales al principio pero a medida que avanza comenzarán a dar resultados erróneos, debido a que es diferente la división de una suma que la suma de una división con otro número.

c) Línea 19: asignando inicio = medio.
Podría formarse un bucle infinito, porque el índice inicio nunca se actualizaría, y el bucle seguiría iterando en el mismo rango de búsqueda una y otra vez, lo que provocaría un bucle infinito. Esto ocurre porque no hay progreso en la búsqueda, y la misma posición medio se selecciona una y otra vez.

d) Idem C.

5.6

a) Sumar dos enteros de N dígitos
Es de orden O(n) debido a que hay que recorrer todos los dígitos de los números.

b) Multiplicar dos enteros de N dígitos
Es de orden $O(n^2)$, esto es debido a que hay que por cada cifra de uno se multiplica la del otro.

c) Dividir dos enteros de N dígitos.
Es de orden O(1) debido a que no hay que recorrer nada, solo depende de las cifras.

b) Si el algoritmo es $O(N \log N)$, entonces el tiempo de ejecución aumenta en un factor logarítmico cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño n será:

T. de ejecución $= (n/100) \times 0{,}5\ ms \times \log(n/100)$

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de n tal que el tiempo de ejecución sea igual a 60 segundos (o 60000 ms). Utilizando una calculadora en línea encontramos que n es aproximadamente igual a 259200.

c) Si el algoritmo es cuadrático, entonces el tiempo de ejecución aumenta en un factor cuadrático cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño n será:

T. de ejecución $= (n/100)^2 \times 0{,}5\ ms$

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de n tal que el tiempo de ejecución sea igual a 60 segundos (o 60,000 ms). Despejando n:

$(n/100)^2 \times 0{,}5\ ms = 60{,}000\ ms$

$n = \sqrt{(60{,}000\ ms / 0{,}5\ ms)} \times 100$

$n = 34641$

Por lo tanto, con un algoritmo cuadrático que tarda 0,5 ms en procesar una entrada de tamaño 100, puede procesar una entrada de tamaño 34641 en un minuto.

d) Si es cúbica, entonces el tiempo de ejecución aumenta en un factor cúbico cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño n será:

T. de ejecución: $(n/100)^3 \times 0,5$ ms

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de n tal que el tiempo de ejecución sea igual a 60 segundos (o 60000 ms). Despejando n:

$(n/100)^3 \times 0,5\text{ms} = 60000$ ms

$n = \sqrt{(60000\text{ms} / 0,5\text{ms})} \cdot 100$

$n = 4932$

5.12

N	$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
10	0,000103	0,00045	0,00066	0,00034
100	0,47015	0,0112	0,00486	0,00063
1000	448,77	1,1233	0,05843	0,00333
10000	NA	111,13	0,68631	0,03042
100000	NA	NA	8,01130	0,29832
1000000	$1X10^9$ T(N)	$1X10^{12}$ T(N)	$2X10^7$ T(N)	$1X10^6$ T(N)

Para entradas muy grandes siempre va a convenir un algoritmo que tenga el más pequeño orden debido a que cada vez va aumentando más y más el tiempo que tardan. Con valores de entrada pequeños casi no se nota la diferencia pero cuando crecen esta diferencia puede ser abismal.

5.13

$37 \rightarrow \frac{2}{N} \rightarrow \sqrt{N} \rightarrow N \log\log N \rightarrow N \rightarrow N \log N \rightarrow N \log N^2 \rightarrow N \log^2 N \rightarrow N^{1.5} \rightarrow N^2 \log N \rightarrow N^2 \rightarrow N^3 \rightarrow 2^{\frac{N}{2}} \rightarrow 2^N$

Tanto $2^{\frac{N}{2}}$ como 2^N crecen a la misma velocidad, tambien $N^2 \log N$ y N^2, y $N \log N$ con $N \log N^2$.

5.14

Fragmento 1

```
for (int i=0; i<n; i++)   O(N)        O(N) + O(1) = O(N)
    suma++;               O(1)        O(N)
```

Fragmento 2

```
for (int i=0; i<n; i++)   O(N)   } O(N^2)   O(N^2) + O(1)
    for (int j=0; j<n; j++) O(N) }
        suma++;           O(1)        O(N^2)
```

Fragmento 3

```
for (int i=0; i<n; i++)   O(N)        O(N) + O(N) + O(1) + O(1)
    suma++;               O(1)
for (int j=0; j<n; j++)   O(N)        O(N)
    suma++;               O(1)
```

Fragmento 4

```
for (int i=0; i<n; i++)          O(N)   } O(N^2)
    for (int j=0; j<n*n; j++)    O(N)   }
        suma++;                  O(1)
                                        O(N^2) + O(1)
```

Fragmento #5

```
for (int i=0; i<n; i++) O(N)
    for (int j=0; j<1; j++) O(N)
    suma++; O(1)
```

$O(N^2)$ $O(N^2)$

$O(N^2) + O(1)$

Fragmento #6

```
for (int i=0; i<n; i++) O(N)
    For (int j=0; j<n*n; j++) O(N)
        For (int k=0; k<j; k++) O(N)
        suma++; O(1)
```

$O(N^3)$

$O(N^3)$

$O(N^3) + O(1)$

Código	1	2	3	4	5	6
Suma con N=10	10	100	20	1000	45	49500
	0ms	0ms	0ms	0ms	0ms	0ms
Suma con N=1000	1000	1000000	2000	1000000000	499500	882236160
	0ms	2ms	0ms	15ms	1ms	1251ms

5.15

```
for (int i=1; i<n; i++) O(N)
    for (int j=0; j<1*i; j++) O(N)
        if (j%i==0) O(1)
            for (int k=0; k<j; k++) O(N)
            suma++; O(1)
```

$O(N^3)$

$O(N^3)$

$O(N^3) + O(1) + O(1)$

Suma con N=1000: 29656666/1486 ms

Suma con N=10: 870/0 ms

5.16

Sigue el gráfico de 2^n, por lo que es de este orden. En el día N basta con sustituir n en 2^n