

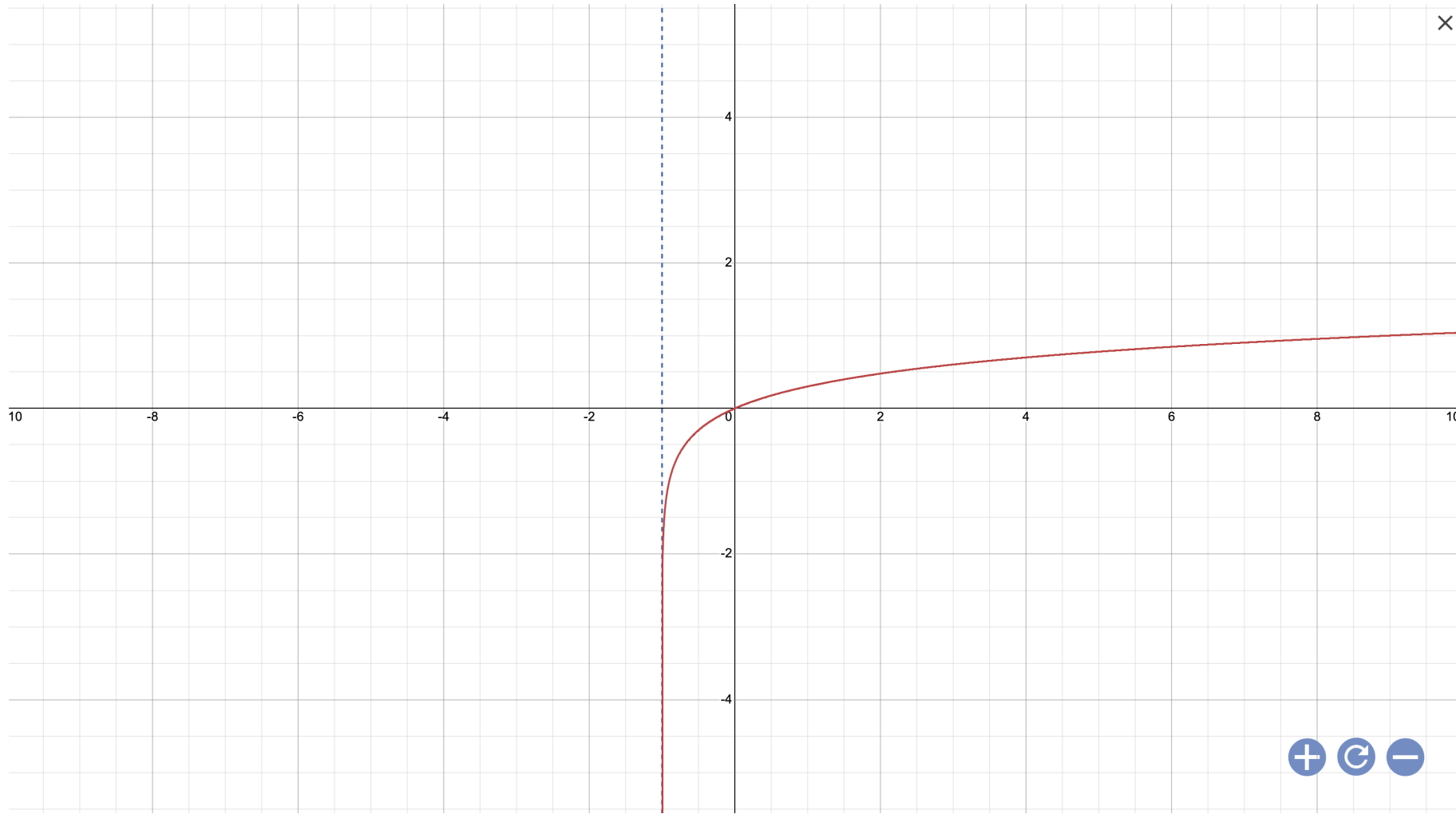
Estructuras de Datos

Notación O grande (Big-O)

Adriana Collaguazo Jaramillo, Mg.

Qué es?

En matemáticas, ayuda a describir el comportamiento de una función al límite, cuando el argumento tiende a un valor específico.



Qué es?

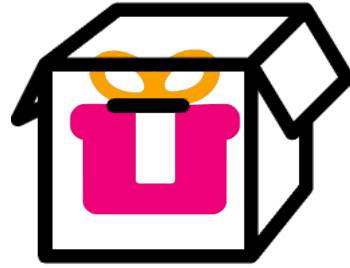
En computación, ayuda a describir el comportamiento de un algoritmo



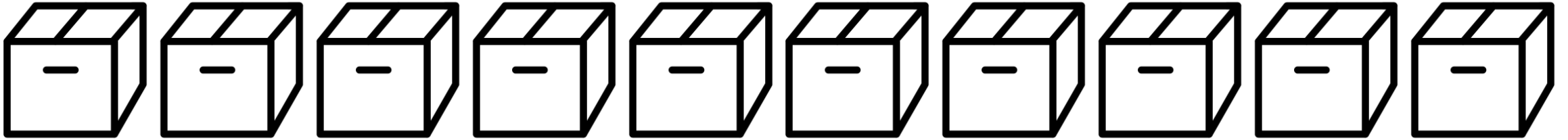
Análisis útil para grandes entradas

Scenario

Buscando un regalo



n cajas:



Si se abre una caja a la vez...

El análisis **O** grande se pregunta:

¿En el **peor de los casos**, cuántas cajas hay que abrir para encontrar el regalo?

Respuesta

En el peor de los casos, el regalo está en la última caja (la caja n)

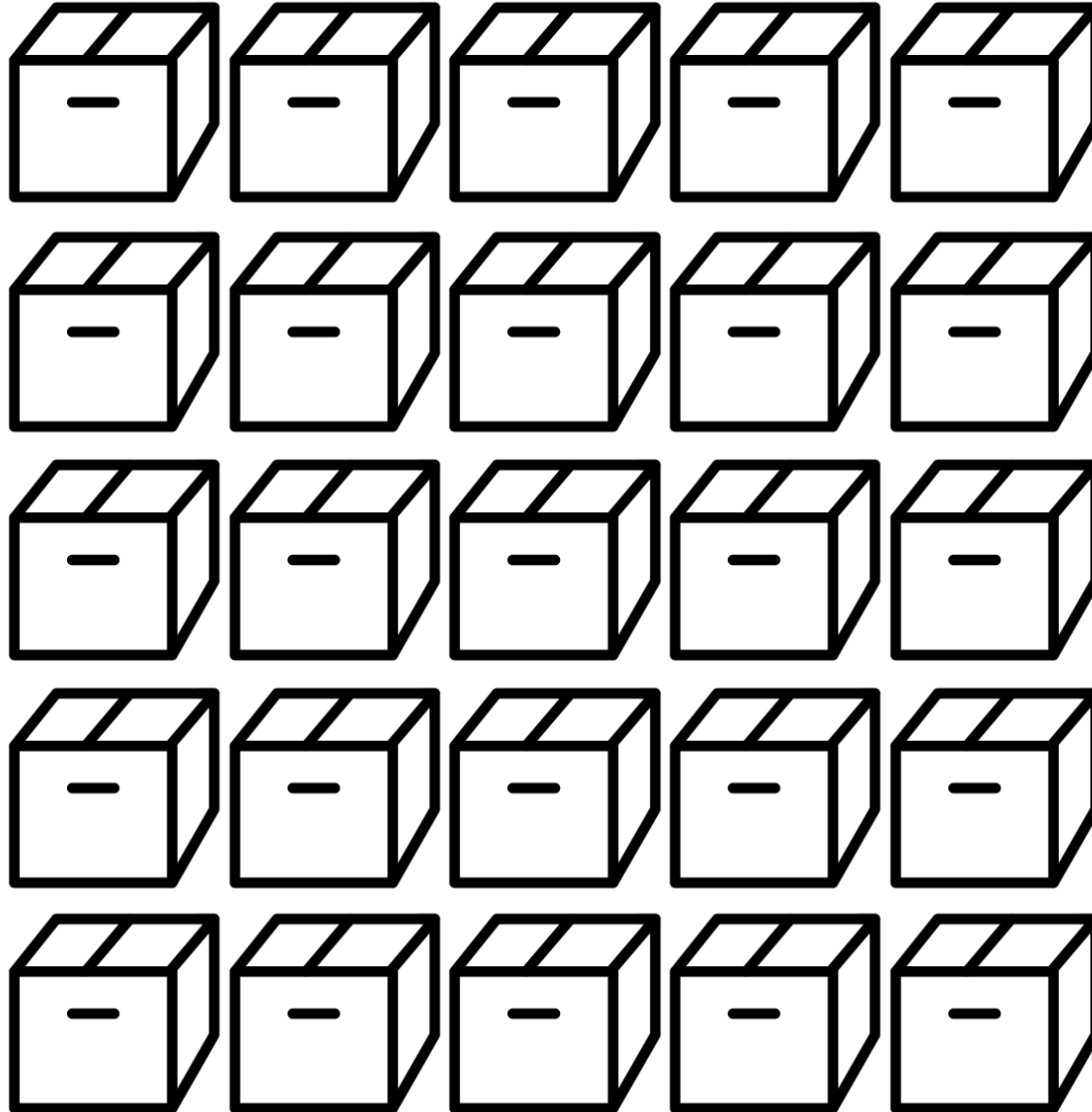
Dada una entrada de tamaño n , este algoritmo tiene una complejidad $O(n)$

La complejidad de este algoritmo es **lineal**

Es decir, el número de operaciones necesarias para resolver el problema es una **función lineal del tamaño de la entrada**

La letra O (mayúscula) se lee “*en el peor de los casos*”

¿Y si usamos una matriz de $n \times n$ cajas?



Abriendo una caja a la vez

Complejidad $O(n^2)$

Complejidad cuadrática

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7

1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

¿7 == 14?

¿7 < 14?

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7

1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

¿7 == 6?

¿7 < 6?

¿7 > 6?

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7

1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

¿7 == 8?

¿7 < 8?

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7

1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Otras Complejidades

Ejemplo: Búsqueda binaria

Buscando el número 7



1	3	4	6	7	8	10	13	14	18	19	21	24	37	40	45	71
---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

¿7 == 7?

El algoritmo divide los datos y descarta parte de ellos en cada paso

Complejidad logarítmica: $O(n \log (n))$

Complejidad Constante

Algoritmo para calcular la suma $a + b + c$

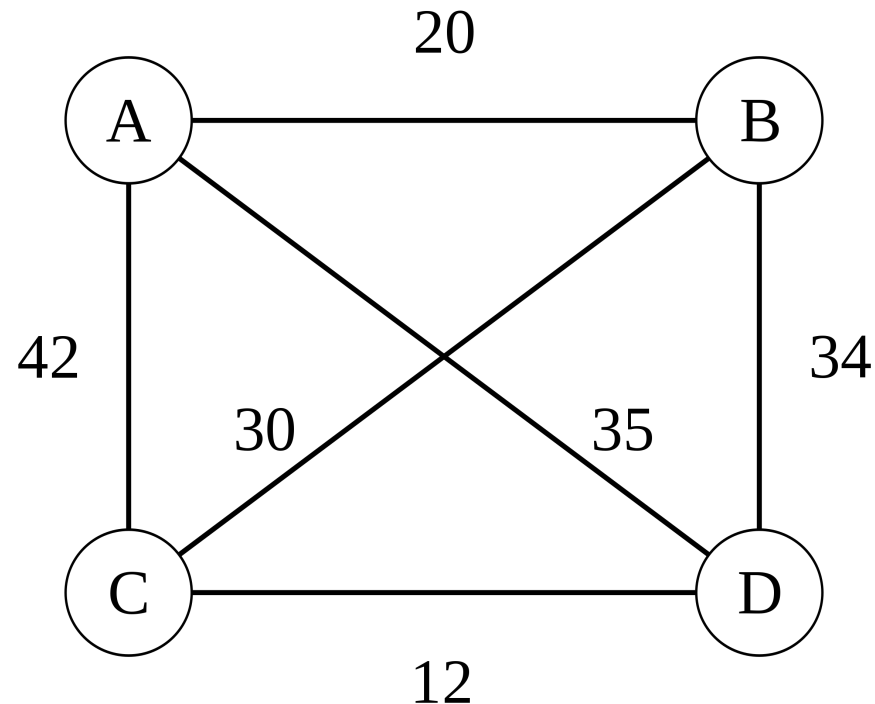
Independientemente de los valores de a , b y c , el algoritmo se ejecuta en el mismo tiempo

$O(1)$: complejidad constante

Complejidad Factorial

Problema del agente viajero:

Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?



Complejidad Factorial

Fuerza bruta (probar todas las posibilidades) con diez ciudades:

3'628.800 operaciones

Si tenemos 28 capitales Europeas

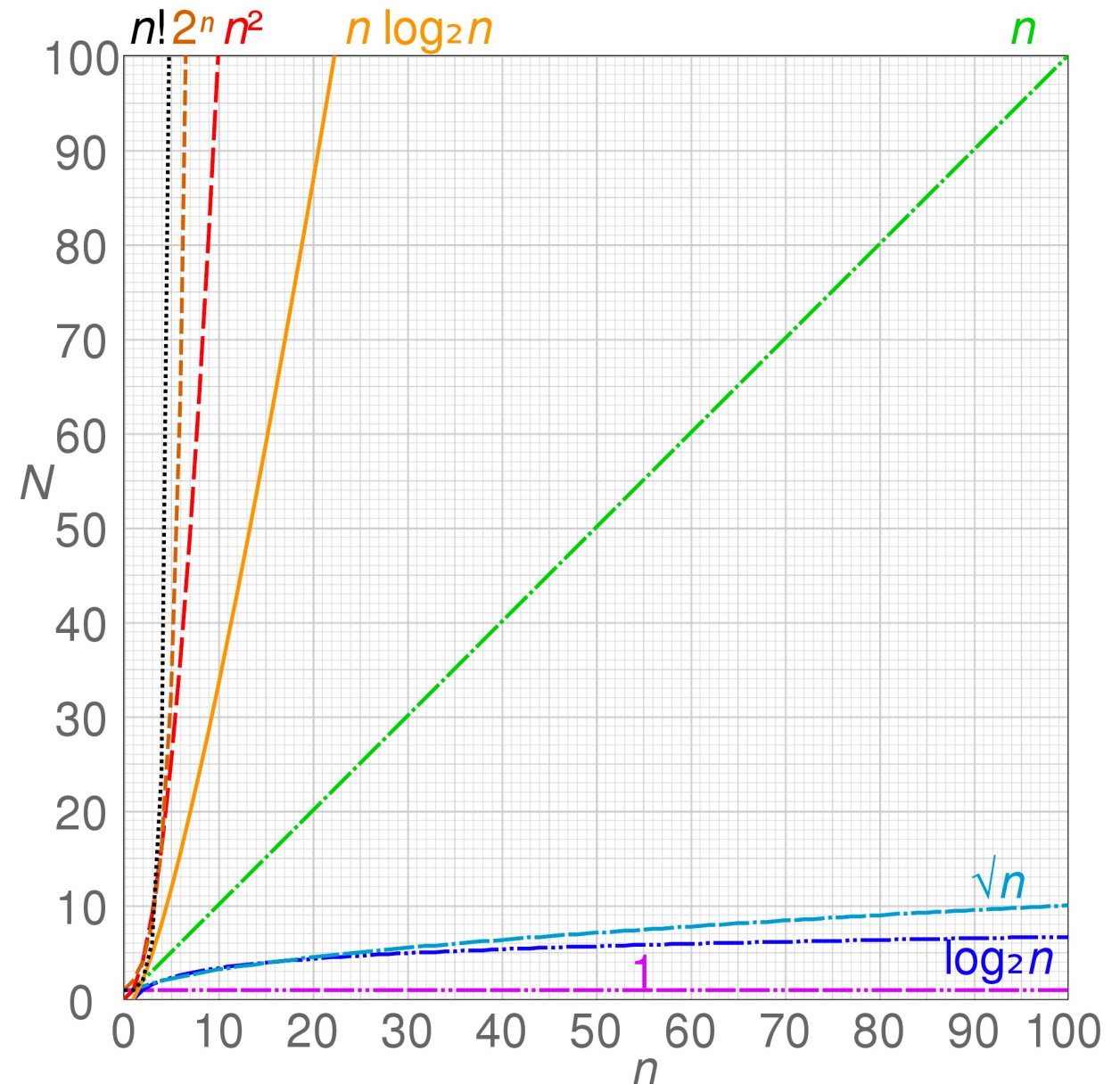
Aplicando fuerza bruta:

3.048.883.400.000.000.000.000.000.000.000.000.000 operaciones

$O(n!)$: complejidad factorial

En Resumen

O: Describe cómo cambia el tiempo de ejecución de un algoritmo en función del tamaño de la entrada



¿Para qué sirve?

- Dispositivos con recursos limitados
- Sistemas cuya eficiencia es crítica: Sistemas de tiempo real
- Ayuda a elegir con qué algoritmos trabajar

¿Por qué es importante en este curso?

Algunos algoritmos son mejores que otros

Distintas operaciones en un mismo TDA tienen distinta **eficiencia**

La misma operación en distintos TDAs puede tener distinta **complejidad**

En el examen, podría ocurrir que usted deba implementar algoritmos de una complejidad específica.

Ejercicio

```
function something() {  
    doStep1(); //  $O(a)$   
    doStep2(); //  $O(b)$   
}
```


Ejercicio

```
function minMax1(array) {  
  min, max ← NULL  
  for each e in array  
    min = MIN(e, min)  
  for each e in array  
    max = MAX(e, max)
```

```
function minMax2(array) {  
  min, max ← NULL  
  for each e in array  
    min = MIN(e, min)  
    max = MAX(e, max)
```

Ejercicio

```
int intersectionSize(arrayA, arrayB){  
    int count = 0  
    for a in arrayA {  
        for b in arrayB {  
            if a == b {  
                count = count + 1  
            }  
        }  
    }  
    return count  
}
```

Ejercicio

```
function whyWouldIDoThis(array){
```

```
    max = NULL
```

```
    for each a in array {
```

```
        max = MAX(a, max)
```

```
    }
```

```
    print max
```

```
    for each a in array {
```

```
        for each b in array {
```

```
            print a, b
```

```
        }
```

```
    }
```

```
}
```

Tema de Examen

Un algoritmo ordena los n elementos de una estructura de datos en tres pasos. La tabla mostrada a continuación detalla el tiempo de ejecución de cada uno de estos pasos en cinco implementaciones distintas del algoritmo. Usando la notación O grande, escriba en la fila inferior de la tabla la complejidad de cada una estas implementaciones.

	Implementación 1	Implementación 2	Implementación 3	Implementación 4	Implementación 5
Paso 1	$\log(n)$	$n \log(n)$	$37n$	$1000n^2$	2^{10}
Paso 2	1000	$15n$	$n \log(n^2)$	$16n$	3^5
Paso 3	$27 \log(n)$	$0.002n^2$	$5000 \log(n)$	2^n	1000000
O					