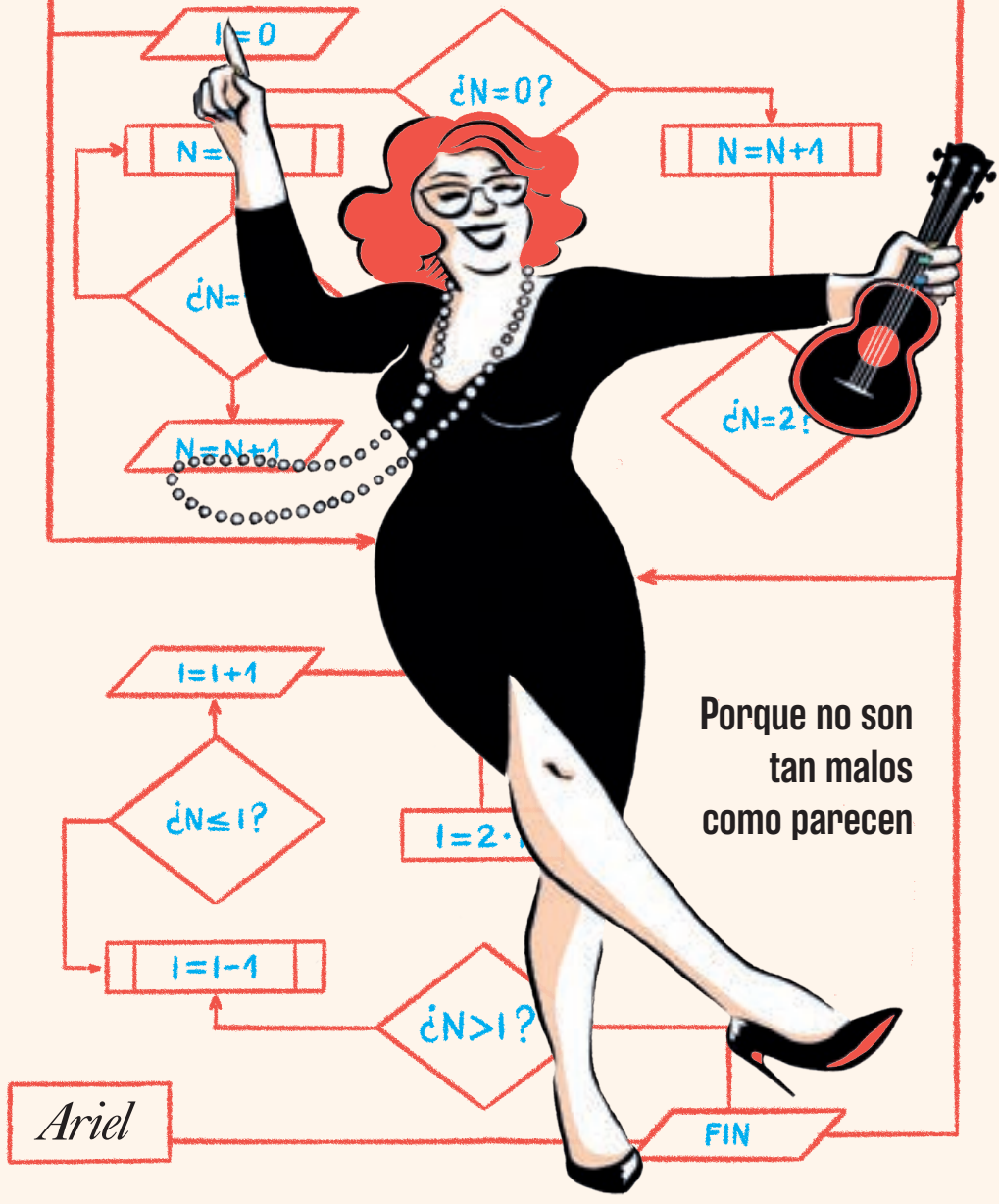
CLARA GRIMA

CON ALGORITMOS Y A LO LOCO



Porque no son tan malos como parecen

Ariel

CLARA GRIMA

CON ALGORITMOS Y A LO LOCO

Porque no son tan malos como parecen

Ariel

Primera edición: junio de 2025

Avda. Diagonal, 662-664, 08034 Barcelona
Editorial Ariel es un sello editorial de Planeta, S. A.
www.ariel.es
www.planetadelibros.com

ISBN: 978-84-344-3898-9
Depósito legal: B. 7.401-2025

Impreso en España

Índice

1

Sopa de ganso, algoritmos neperianos y otras recetas básicas

No sé si, cuando has leído el título de este primer capítulo, te has asustado, te has reído de medio lado con su pizquita de maldad o has sonreído pensando que no tengo vergüenza. O si, simplemente, no le has hecho el más mínimo caso al mismo, convencida o convencido de que, en mis libros, los títulos de los capítulos son, casi siempre, poco descriptivos.

No lo sé. Ni lo sabré, a no ser que me lo cuentes un día a través de las redes sociales. O si vienes a verme a alguna conferencia o presentación. Pero lo cierto es que, cuando he tecleado el título del capítulo (que, por supuesto, es lo pri-

mero que he escrito de este libro), me he quedado pensando en las posibles reacciones al mismo.

¿Por qué? Porque, hasta donde yo sé, con los cincuenta y cuatro años que tengo mientras escribo esto, no existe el algoritmo neperiano. Lo que sí existe (y bendito sea John Napier, viva Escocia y algunos de sus escoceses) es el logaritmo neperiano, el cual, me atrevo a afirmar sin temor a equivocarme, es una de las herramientas que más ha mejorado nuestras vidas. Pero no hemos venido a hablar de logaritmos. Por ahora.

Dicho esto, puede que, si no conoces el tema, la expresión «algoritmo neperiano» te suene a algo complicado y te hayas podido asustar. Por otra parte, si sabes que no existe tal algoritmo y te encanta descubrir las erratas en libros, puede que te hayas reído con un poco de malicia, pensando que me has pillado un fallo nada más empezar. O bien, si sabes del tema y eres consciente de que durante algún tiempo, no muy lejano, en algunos medios de comunicación era común usar la palabra «logaritmo» para referirse a un «algoritmo», has adivinado que estaba haciendo un chiste con ese hecho.

Sí, no hace muchos años, era común encontrar en los medios titulares como este de 2009, en *El País*, que me llegó a través de Miguel Ángel Morales Medina, autor del mítico blog de matemáticas *Gaussianos*: «Google crea un logaritmo para identificar a sus empleados descontentos».

El titular fue corregido enseguida, es cierto, pero no deja de ser curioso que en 2009, en un medio de la tirada del mencionado, aún no supieran diferenciar entre un algoritmo y un logaritmo.

También fue *Gaussianos*, en 2012, quien detectó que nuestro querido David Trueba mencionaba en su columna en el mismo medio, del 1 de febrero, al «secreto logaritmo de búsqueda» de Google.

Es curioso también, al menos para mí, que la palabra «algoritmo» pasara, en muy poco tiempo, de ser una desconocida para una parte importante de la sociedad (intelectuales

incluidos) a ser una palabra que se puede escuchar en casi cualquier conversación. En una cafetería, en la sala de espera del centro de salud, en la peluquería, en el autobús...

Bueno, como diría el bardo de Avon, *all's well that ends well* («bien está lo que bien acaba»), ¿no? Pues no, no del todo en esta historia. Porque si bien el término «algoritmo» ha pasado del ostracismo al lenguaje coloquial, por el camino se le ha dotado de unos rasgos malignos que, en esencia, no tienen nada que ver con el significado de la palabra. A veces, me hace gracia escuchar a gente hablando de los algoritmos como si estos tuviesen vida, como si fuesen criaturas oscuras y siniestras, cual dementores en el universo de Harry Potter. Pero eso solo me ocurre algunas veces. Cada vez menos, también te digo.

Por eso tienes este libro entre las manos. Porque he decidido hacer una apología de los algoritmos, los cuales no son ni buenos ni malos: son herramientas. Herramientas que nos han ayudado, nos ayudan y nos ayudarán a mejorar nuestro mundo.

Y porque te va a encantar descubrirlos. Si no los conoces ya, claro. Si los conoces, tómate esto como un paseíto por el bulevar de los algoritmos. Un paseo que no pretende ser exhaustivo, porque será eso, un paseo, no una inspección ni una auditoría; y en los paseos nos fijamos en aquello que nos llama más la atención.

¿Qué es un algoritmo?

Vamos a empezar por definir qué es un algoritmo. Aunque, como he dicho unas líneas antes, ya todo el mundo hable de ellos, no sé si todos conocemos su definición.

Mientras que, según el diccionario de la Real Academia Española, un logaritmo es un «exponente al que es necesario elevar una cantidad positiva, llamada base, para que resulte un número determinado», un algoritmo es en su pri-

mera acepción: «Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema».

Y esa es la definición con la que nos quedamos. No es que el diccionario de la RAE sea muy riguroso y actualizado con los términos matemáticos (aún no han incluido la palabra «escutoide», por ejemplo), pero esta definición de algoritmo nos sirve, para empezar.

Si te fijas en la definición anterior, verás que podría ser perfectamente la definición de una receta de cocina: conjunto ordenado y finito de operaciones que permite elaborar una determinada comida. De hecho, estoy casi segura de que, a estas alturas del siglo XXI, ya habías escuchado más de una vez esta analogía entre algoritmo y receta de cocina. No es que esté siendo muy original, lo sé. Es el ejemplo que usamos muchas profesoras y profesores de matemáticas para explicar qué es un algoritmo.

En realidad, un algoritmo debe tener algunas características más, aparte de ser ordenado y finito. En primer lugar, debe estar bien definido. Es decir, cada paso debe estar bien especificado, sin ambigüedades. Y, por supuesto, debe ser general, para poder ser aplicable a problemas similares. Un algoritmo no es un procedimiento que solo se use para resolver un único problema concreto.

Vamos a verlo con un ejemplo muy simple y cotidiano: vamos a escribir un algoritmo para lavarnos el pelo en la ducha. Lo hacemos por pasos, todo muy formal.

— Paso 1: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 2: Meterse bajo la ducha.
— Paso 3: Cerrar el grifo...

Un momento. Ese paso 2 no está, creo, bien definido. Porque no especifica que haya que mojarse el pelo. Pudiera ser que alguien se metiera bajo la ducha, con la cabeza echada hacia atrás, y no se mojara el cabello. Lo reescribimos:

— Paso 1: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 2: Mojarse todo el cabello.
— Paso 3: Cerrar el grifo.
— Paso 4: Abrir el frasco con el champú y depositar sobre la mano una cantidad de tamaño similar a una nuez.
— Paso 5: Extender el champú sobre el pelo mojado y masajearlo.
— Paso 6: Volver al paso 1.

Lo tenemos, ¿no? Es una secuencia bien definida y ordenada. Sí, pero no acaba nunca: tenemos que obligarlo a terminar. ¿Cómo? Pues depende de las veces que te suelas poner champú. Lo habitual son dos veces. Así que podemos escribirlo de la siguiente manera:

— Paso 1: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 2: Mojarse todo el cabello.
— Paso 3: Cerrar el grifo.
— Paso 4: Abrir el frasco con el champú y depositar sobre la mano una cantidad de tamaño similar a una nuez.
— Paso 5: Extender el champú sobre el pelo mojado y masajearlo.
— Paso 6: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 7: Aclarar el cabello hasta que no quede espuma.
— Paso 8: Cerrar el grifo.
— Paso 9: Abrir el frasco con el champú y depositar sobre la mano una cantidad de tamaño similar a una nuez.
— Paso 10: Extender el champú sobre el pelo mojado y masajearlo.
— Paso 11: Abrir el grifo y regular la temperatura del agua a nuestro gusto.

— Paso 12: Aclarar el cabello hasta que no quede espuma.
— Paso 13: Cerrar el grifo.
— FIN.

Con esto ya estaría. Hemos descrito de forma ordenada, clara y finita cómo lavarnos el pelo usando champú dos veces. Está bien, pero podríamos haberlo escrito de forma más compacta, usando un contador que cuente las veces que nos ponemos champú. A ese contador le llamaremos j, por lo de jabón. Y, claro, al principio será igual a 0.

— Paso 1: j = 0.
— Paso 2: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 3: Mojarse todo el cabello.
— Paso 4: Cerrar el grifo.
— Paso 5: Abrir el frasco con el champú y depositar sobre la mano una cantidad de tamaño similar a una nuez.
— Paso 6: Extender el champú sobre el pelo mojado y masajearlo.
— Paso 7: j = j + 1
— Paso 8: Abrir el grifo y regular la temperatura del agua a nuestro gusto.
— Paso 9: Aclarar el cabello hasta que no quede espuma.
— Paso 10: Cerrar el grifo.
— Paso 11: Si j<2 volver al paso 5.
— FIN.

Esta segunda escritura de nuestro algoritmo tiene otra ventaja, además de ser más compacta: si a alguien le gusta ponerse champú tres o cuatro veces, solo tiene que modificar el paso 11, indicando el número de veces que se quiere enjabonar la cabeza.

Además, de esta forma, el algoritmo es general. Sirve para que se lave el cabello casi cualquier persona, salvo excepciones. Los calvos entre otros.

Tenemos, por lo tanto, un algoritmo ordenado, finito, bien definido y general para lavarnos el pelo en la ducha. No puedo asegurarlo, porque cada uno es como es, pero posiblemente este sea el algoritmo que usamos todos para tal menester.

Lo que yo pretendía, sobre todo, era ver con un ejemplo las cuatro características que debe tener cualquier algoritmo: ordenado, finito, definido y general.

Hay otra característica de los algoritmos que, aunque no es estrictamente necesaria cumplirla, sí que es bastante conveniente: la eficiencia. Cuando se diseñan algoritmos, se trata de hacerlos eficientes, lo que significa que queremos que sean óptimos en términos de tiempo y recursos. Y a esto último, a conseguir algoritmos correctos (con las cuatro características señaladas) y eficientes, es a lo que se dedican muchas horas de investigación.

Más rápido, por favor

Tendemos a pensar que los ordenadores actuales resuelven cada problema de forma casi inmediata, que realizan cualquier cálculo instantáneamente. Pero esta idea dista de ser cierta. Aunque es indudable que la velocidad de un simple portátil (incluso de un móvil) actual resultaba casi impensable hace unas cuantas décadas, hay que tener cuidado al encomendarle una tarea, puesto que puede que, si no somos cuidadosos, se sobrepase el tiempo del que disponemos para obtener una respuesta.

Voy a tratar de ilustrarlo con un ejemplo que, creo, es fácil de entender.

Vamos a pensar que somos los encargados de diseñar las rutas de una empresa de paquetería y queremos calcular el recorrido óptimo, el más corto posible, para hacer entregas de los paquetes en cinco direcciones distintas.

Una forma de dilucidar dicho recorrido sería, si conocemos el tiempo que tardamos entre cada par de puntos, cal-

cular el tiempo total para cada ordenación distinta de los cinco puntos y quedarnos con el mejor de dichos tiempos.

Si tenemos que entregar paquetes en las direcciones **D1**, **D2**, **D3**, **D4** y **D5** saliendo y volviendo a nuestro almacén **A**, un posible recorrido sería el que nos define la ordenación (**D1**, **D2**, **D3**, **D4**, **D5**), es decir: **A-D1-D2-D3-D4-D5-A**. La longitud de este recorrido sería (si llamamos *d(P,Q)* a la distancia entre los puntos *P* y *Q*):

d(A, D1) + d(D1, D2) + d(D2, D3) + d(D3, D4) + d(D4, D5) + d(D5, A)

Otro recorrido posible sería, por ejemplo, el correspondiente a la ordenación (**D2**, **D4**, **D5**, **D1**, **D3**), es decir: **A-D2-D4-D5-D1-D3-A**. Este segundo recorrido tendría de longitud:

d(A, D2) + d(D2, D4) + d(D4, D5) + d(D5, D1) + d(D1, D3) + d(D3, A)

Evidentemente, siguiendo esta estrategia encontraremos la ruta óptima, puesto que estaríamos examinando todas las posibles y no son tantas. Concretamente, si comenzamos y terminamos siempre en el mismo punto (nuestro almacén **A**) cada una de las posibles ordenaciones de los otros cinco puntos nos dará una ruta. ¿Cuántas ordenaciones distintas son posibles? Al número de ordenaciones posibles se le conoce como «permutaciones de cinco elementos» y es **5 × 4 × 3 × 2 = 120** posibles rutas (lo que las matemáticas y los matemáticos escribimos como **5!** y leemos como **factorial de 5**). Y es cierto que cualquier ordenador realiza ya hoy en día los cálculos necesarios para resolver este problema en casi un instante, en un tiempo que no percibimos con los sentidos humanos. En este caso, necesitamos sumar seis cantidades para cada ruta y realizar comparaciones, así que el número total de operaciones no llega ni a mil. Pero si nuestra ruta pasa por un número mayor de puntos la cosa varía. Incluso si despreciamos (cosa que es peligrosa) el número de operaciones que necesitamos para calcular el tiempo de cada ruta, el número de posibles rutas puede ser muy grande, realmente grande, enorme. Por ejemplo, supongamos que tenemos que visitar 33 puntos, 33 direcciones, desde nuestro almacén. En ese caso, el número de posibles rutas es 33!, es decir, **33 × 32 × 31 × 30 × ... × 3 × 2**.

Puestos a suponer, supongamos que tenemos a nuestra disposición el ordenador más potente del momento, uno que realiza más de 1.000.000.000.000.000.000 operaciones por segundo, y supongamos que calcula el tiempo de cada ruta en una operación (en realidad, sabemos que son bastantes más, aproximadamente una operación por cada punto a visitar). Pues aun así necesitaríamos un tiempo de veinte. Alguien dirá que veinte no es tanto, pero es que no he dicho las unidades. ¿Por qué apostarías? ¿Veinte segundos? ¿Veinte minutos? ¿Veinte horas? Desde luego, si se trata de este último caso, veinte horas, puede no ser viable este método, ya que cada día nuestra flota estaría inmovilizada du-

rante esas horas hasta que decidamos qué ruta asignar a cada vehículo. Pero no es el caso, no son veinte horas, sino algo peor, y tampoco son días o años o siglos, sino la edad del universo. Con el computador más rápido de la actualidad necesitaríamos veinte veces la edad del universo para realizar esos cálculos. Nuestra empresa de transporte se iría a la quiebra. Parece evidente que nuestro algoritmo de analizar todas las rutas posibles es una mijita lento, poco eficiente. Más adelante, en este libro, veremos cómo encontrar una buena solución en tiempo razonable.

Pero sigamos por ahora con este ejemplo. En él hay unas cuantas ideas que son fundamentales, aunque obvias. En primer lugar, los ordenadores no son todopoderosos, no realizan sus cálculos al instante. Y en segundo lugar, y en ello nos vamos a concentrar a continuación, no es lo mismo realizar una tarea para 5 puntos que para 33: el tamaño del problema (de lo que llamamos la entrada del algoritmo) importa. Si juntamos estas dos ideas llegamos a la conclusión de que no vale resolver un problema de cualquier forma, sino que hemos de intentar que el algoritmo que diseñemos sea lo más eficiente posible. Y esto se traduce en que realice el mínimo número de operaciones posibles. Si para resolver un problema diseñamos diferentes algoritmos que lo resuelvan, al comparar dos de estos algoritmos nos quedaremos con el que sea más rápido, es decir, con el que realice menos operaciones para una entrada de tamaño dado.

Voy a tratar de ilustrarlo con otro ejemplo. Nos dan una lista de números y nos piden que la ordenemos de menor a mayor. Como ya sabemos por el ejemplo de las rutas, esta es una tarea que dependerá de la cantidad de números que nos den: cuanto más grande sea la lista, más operaciones necesitaremos. Pero también dependerá del método que diseñemos para ordenarla. Por ejemplo, podemos dar todas las permutaciones posibles (todas las ordenaciones) de los números de la lista, como hemos hecho antes, y ver cuál de ellas es en la que aparecen todos los números ordenados.

Pero ya sabemos que, salvo que la lista tenga muy pocos números, este método, al que llamaremos algoritmo $\mathbf{A_1}$, es inviable. Así que hemos de intentar buscar una idea mejor.

Algo simple, pero que funciona bien, es tomar los dos primeros números de la lista y ordenarlos, y después comparar el tercero con el primero y el segundo para determinar su posición, y lo mismo con el cuarto, etcétera.

Por ejemplo, si nos dan la lista {4,1,5,2,3} comenzamos comparando el 4 con el 1.

¿Es 4 menor que 1? Como la respuesta es no, intercambiamos sus posiciones. Así, estos dos quedarían ya ordenados: {1,4}

Ahora es el momento de colocar al 5 en su sitio.

¿Es 5 menor que 1? No. El 5 se coloca después del 1 y nos queda {1,5,4}. ¿Hemos terminado de colocar el 5? No, tenemos que compararlo con el número a su derecha. ¿Es 5 menor que 4? No. Los intercambiamos y llegamos a {1,4,5}, paso final por ahora porque no quedan números a la derecha del 5.

Es el turno del 2. ¿Es 2 menor que 1? No. Luego el 2 entra en la lista después del 1: {1,2,4,5} ¿Hemos terminado de colocar el 2? No, tenemos que compararlo con el número a su derecha. ¿Es 2 menor que 4? Sí. Ea, pues ya está bien colocado.

Nos falta colocar el 3 en la lista. ¿Es 3 menor que 1? No. Lo ponemos después del 1: {1,3,2,4,5} ¿Hemos terminado de colocar el 3? No, tenemos que compararlo con el número a su derecha. ¿Es 3 menor que 2? No, los intercambiamos: {1,2,3,4,5}. ¿Hemos terminado de colocar el 3? No, tenemos que compararlo con el número a su derecha. ¿Es 3 menor que 4? Sí. Hala, ya hemos terminado.

Este método, o algoritmo $\mathbf{A_2}$, para una lista con cinco números en vez de necesitar $\mathbf{5!}$ operaciones, realiza, como máximo, en el caso más desordenado, $\mathbf{5^2}$ operaciones. Lo cual es mucho, pero también es muchísimo mejor. Porque para 33 números solo necesitaría $\mathbf{33^2 = 1.089}$ operaciones,

que son muchísimas menos que **33!** (que es del orden de 10^{35}, un 1 con 35 ceros detrás).

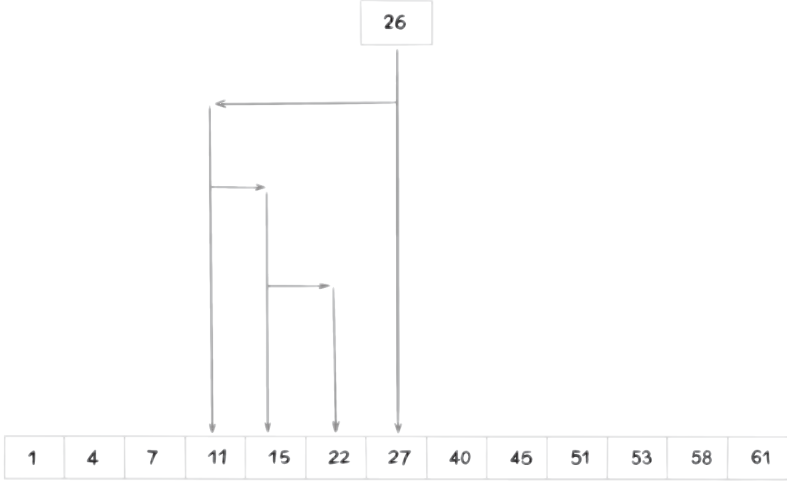
Algo importante, que en realidad ya estaba implícito en lo dicho anteriormente, es que el número de operaciones depende del número de elementos de la lista a ordenar. En otras palabras: en un algoritmo el número de operaciones depende del tamaño de la entrada (y en muchas ocasiones no solo del tamaño, pero eso es harina de otro costal).

¿Se puede mejorar el algoritmo A_2?

Pues sí, con una ligera modificación. Con este algoritmo, si tenemos ordenados ya unos cuantos números de la lista, al tratar de colocar a uno nuevo en su sitio lo tenemos que comparar con el primero, después el segundo, y así sucesivamente hasta encontrar su posición. Pero ¿y si empezamos comparándolo con el que se halla a mitad de la lista ya ordenada, para decidir si está en la mitad de los más pequeños o en la mitad de los más grandes? Repitiendo esta idea sucesivamente, siempre partiendo por la mitad, se puede comprobar que, en general, se necesitan menos operaciones para insertar un nuevo número en la lista en su posición correcta.

Veámoslo con otro ejemplo, que siempre es más visual. Supongamos que tenemos la siguiente colección de números ya ordenados: {1, 4, 7, 11, 15, 22, 27, 40, 45, 51, 53, 58, 61} y tenemos que insertar, en su sitio, el 26. Si lo comparamos con 1, 4..., vamos a necesitar realizar siete comparaciones (desde el 1 hasta el 27). Pero si lo comparamos directamente con el elemento central de la lista (como tenemos trece números, el que ocupa la mitad de la tabla es el séptimo, que es precisamente el 27), como el 26 es menor, concluimos que debe estar en la mitad de los más pequeños: {1, 4, 7, 11, 15, 22, 27}. Nos olvidamos del resto (de la mitad de los más grandes) y comparamos el 26 con el que ocupa la mitad de esa nueva lista, es decir, con el que está en tercer lugar,

el 11. El 26 es mayor que el 11 y, por lo tanto, debe estar entre el 11 y el 27: {11, 15, 22, 27}. Como el número de elementos entre el 11 y el 27 es par, escogemos uno de los dos centrales, por ejemplo, el 15 (también es mala suerte, pues si hubiéramos escogido el otro tendríamos una operación menos, pero eso no se sabe *a priori* cuando se diseña el algoritmo). El 26 es mayor que el 15, por lo que, entonces, toca compararlo con el 22 para encontrar su posición. Hemos realizado solo cuatro comparaciones y eso que el número que hemos escogido para nuestro ejemplo es el peor para este método. En la figura siguiente se da un esquema de lo que acabamos de explicar.



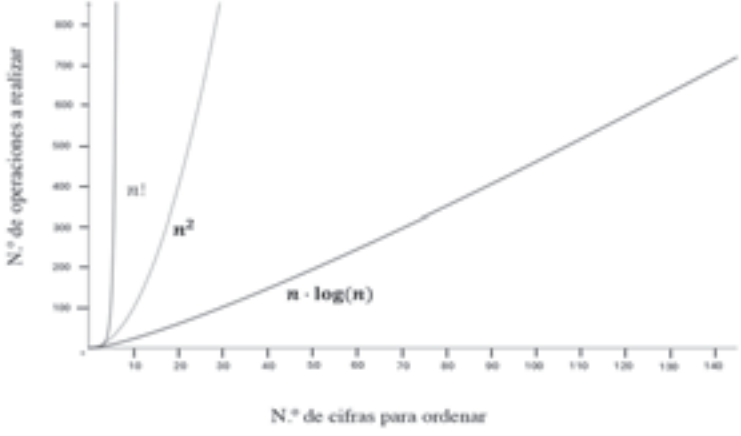
Cada flecha vertical indica con qué número comparamos cada vez y las horizontales marcan la mitad con la que nos quedamos.

Se puede comprobar así que, en general, usaremos menos operaciones con este método, al que llamaremos algoritmo $\mathbf{A}_3$.

Pues bien, recapitulando tenemos que con el algoritmo $\mathbf{A}_1$ (el de considerar todas las posibles ordenaciones) para ordenar $\boldsymbol{n}$ números necesitamos $\boldsymbol{n!}$ operaciones; con el algo-

ritmo $\mathbf{A_2}$, para ordenar n números necesitamos n^2 operaciones, y (sin entrar en los detalles técnicos, que no es este el sitio ni el momento, pero confía en mí, que soy una señora con gafas) con el algoritmo $\mathbf{A_3}$ para ordenar n números necesitamos $n \times \log(n)$ (donde $\log(n)$ representa al logaritmo de n).

Comparemos ahora estos tres algoritmos usando la gráfica de la siguiente figura. El eje horizontal muestra el número de números del conjunto a ordenar y el vertical el número de operaciones para $n!$, n^2 y $n \times \log(n)$. En dicha imagen, cuanto más baja es la curva, menos operaciones se realizan. Como ves, esa extraña $n \times \log(n)$ es la mejor (hasta el momento).



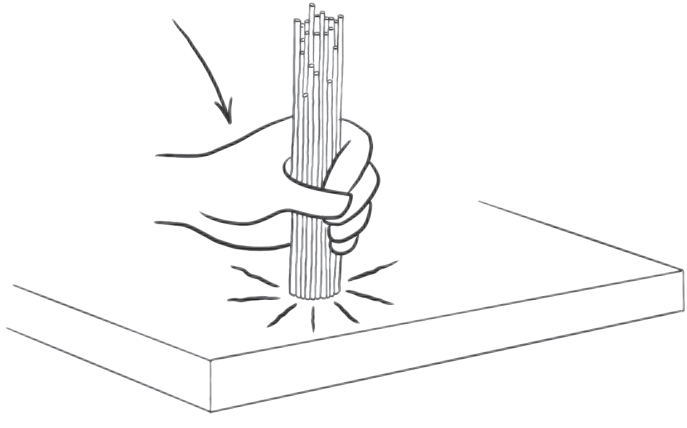
Resumiendo, para comparar dos algoritmos que resuelvan la misma tarea, contaremos el número de operaciones (aproximadamente) que realizamos con cada uno de ellos y nos quedaremos, lógicamente, con el que menos operaciones realice.

Ahora bien, ¿ese número de operaciones no dependerá de la máquina con la que las realicemos? Sorprendentemente no: las operaciones solo dependen del algoritmo y de la entrada. Sí que es cierto que los modernos ordenadores pueden realizar muchas operaciones a la vez, pero esto no nos ahorrará ninguna operación, sino que solo acortará el tiempo de realización.

Bueno, lo que acabamos de decir no es del todo cierto: el número de operaciones también puede depender de la máquina. Estas comparaciones dos a dos son propias de los ordenadores tal y como los conocemos, pero existen otro tipo de «ordenadores» en los que se pueden realizar menos operaciones o, mejor dicho, otras operaciones distintas que no se pueden llevar a cabo en un PC normal, ni siquiera en un supercomputador muy avanzado.

Existe, de hecho, un «ordenador» muy específico para ordenar números que podemos construir nosotros mismos y que requiere menos de $n \, x \, log(n)$ operaciones. Los elementos que necesitamos son: un metro, una superficie plana, como una mesa, por ejemplo, un rotulador y un paquete de fetuchini (como los espaguetis pero planitos y bastante anchos).

¿Cómo usar este «ordenador» para ordenar números? Supongamos, como antes, que nos dan una lista L de n números que tenemos que ordenar. Tomamos el primero de la lista y un fetuchini; con el metro medimos una longitud sobre el fetuchini igual al número que hemos cogido de la lista; cortamos el fetuchini por esa longitud y escribimos el número sobre su superficie plana. Realizamos esa operación para cada número de la lista L; después tomamos todos los fetuchinis con nuestras manos y los igualamos por abajo usando la mesa, tal y como muestra la figura.



Ya los tenemos ordenados: bastará con ir sacándolos uno a uno en orden. De esta forma básicamente hemos realizado del orden de n operaciones que es mejor que el $n \times log(n)$ que se podía conseguir en un ordenador convencional (y que es óptimo en dicho caso).

Ya, ya sé que este último algoritmo solo sirve para jugar un rato y no para la vida cotidiana, pero es simpático, ¿no?

Aunque, como he dicho al principio de este capítulo, vamos a darnos un paseo por la historia de los algoritmos y tratar de explicar el sabor de algunos de ellos, me gustaría terminar esta introducción reivindicando a la matemática británica que nos enseñó a escribir los algoritmos para que lo entendiesen las máquinas, las computadoras.

Estoy hablando, sí, de Ada Lovelace.

La encantadora de números

Ada Lovelace es reconocida mundialmente como la primera persona que escribió un algoritmo para que lo pudiera entender una máquina. Pero no como la primera persona que diseñó un algoritmo, ojo. Las humanas y los humanos hemos usado los algoritmos desde casi el principio de nuestra existencia. La agricultura, por ejemplo, se puede considerar un algoritmo y es una práctica ancestral. Digo esto porque algunas veces nos venimos arriba atribuyendo méritos y se nos va de las manos. Y exagerar es tan peligroso y dañino como mentir. En mi opinión, claro.

Nuestra querida Ada, en realidad, se llamó al nacer, en 1815, Augusta Ada Byron. A lo mejor te suena el apellido Byron y sí, efectivamente, Ada era hija del famoso poeta inglés lord Byron (George Gordon Byron), considerado uno de los más grandes poetas británicos y una de las principales figuras del romanticismo. Pero la verdad es que el señor romántico influyó muy poquito en nuestra amiga, porque el lord se separó de la madre de Ada a los dos meses de nacer ella.

Así que fue su madre, Annabella Milbanke, la responsable de la exquisita educación que recibió Ada. Aunque menos conocida que el padre de la criatura, Annabella fue una mujer muy comprometida con distintas causas sociales que luchó, entre otras metas, por abolir la esclavitud. Si el padre de Ada fue un amante de las palabras, Annabella fue siempre una enamorada de las matemáticas y su lógica. Según cuentan, por esta razón, su marido la llamaba «mi princesa de los paralelogramos». ¿En serio, George? Tú, que eras una de las principales figuras del romanticismo, ¿no encontraste algún objeto matemático menos soso que un paralelogramo para piropear a tu mujer? En fin. Esta pasión de Annabella por las matemáticas y la astronomía y la mala experiencia con el señor poeta influyeron en ella a la hora de elegir la educación para su única hija. Intentando alejar a su niña de la poesía y la locura que parecía provocar en los poetas, según ella, puso a Ada en manos de los mejores matemáticos y científicos de la época. Entre ellos, el mismísimo Augustus De Morgan o la mismísima Mary Somerville. Esta última ha sido otra mujer silenciada tradicionalmente en los libros de historia de las matemáticas a pesar de que fue una impulsora de las mismas en Inglaterra en el siglo XIX. Somerville fue famosa por sus libros, que popularizaron conceptos científicos, incluyendo la mecánica celeste y la física. Sin duda, su labor como divulgadora influyó notablemente en los avances de aquella época.

Fue precisamente en casa de Mary Somerville donde Ada conoció a Charles Babbage, un matemático e ingeniero pio-

nero de la computación. Babbage no fue estrictamente un profesor formal de Ada, pero, desde luego, sí que influyó de manera decisiva en su trabajo.

Babbage le habló, en 1833, de su proyecto de máquina diferencial, una especie de calculadora mecánica, y Ada quedó fascinada por la idea inmediatamente, puesto que, además de la ciencia, era una apasionada de la ingeniería. Solo con trece años ya había escrito un libro sobre las propiedades de las alas de los pájaros, con el propósito de entender el «mecanismo», descifrar el equilibrio exacto entre el tamaño de sus alas y el peso de su cuerpo y poder construir alas para humanos. El libro, lleno de ilustraciones, se llamó *Flyology* (en castellano sería «Vuelología»).

Ni Ada llegó a construir sus alas ni Babbage construyó nunca su máquina diferencial. Pero años más tarde ideó otra máquina, la máquina analítica, que ya podríamos entender como una precursora de las computadoras, de los ordenadores, aunque tampoco se llegó a construir por falta de inversión e interés. Sobre todo, de lo segundo.

El caso es que Charles Babbage presentó su proyecto de máquina analítica en una serie de conferencias en Turín en 1840 y, un par de años más tarde, un matemático italiano que había asistido a dichas conferencias, Luigi Federico Menabrea, publicó un artículo, en francés, explicando el funcionamiento de la misma. Y ahí llega nuestra amiga Ada, en 1843, a traducirlo al inglés. Pero no solo eso, sino que en aras de explicar mejor el funcionamiento y, sobre todo, las distintas posibilidades (más allá de simples cálculos) que ofrecía la máquina analítica, Lovelace añadió unas notas al artículo de Menabrea. En una de ellas, la nota G (las ordenó alfabéticamente), propuso la escritura de un algoritmo para que la máquina calculase una secuencia de números conocidos como los números de Bernoulli. Esta nota es aceptada por la comunidad matemática internacional como el primer algoritmo escrito para una computadora. Aunque aún no existían ordenadores, claro. Ni siquiera la máquina analítica de Babbage.

No deja de ser rompedor que en pleno siglo XIX, siendo ya madre de tres hijos (el más pequeño nació en 1839), una mujer trabajara y publicara sus resultados en matemáticas. Sin duda, la vida de Lovelace da para una película o una serie (¿algún productor en la sala?), pues fue una adelantada a su época y una mente muy brillante que, desafortunadamente, perdimos demasiado pronto. Ada murió de cáncer a los treinta y seis años. Según lo que han escrito algunos autores sobre ella, a pesar del empeño de su estricta y religiosa madre por que estudiara ciencias y así apartarla de la poesía y de la incontinencia moral que esta provocaba en los y las poetas, Ada también tuvo tiempo (en la década de los cuarenta) de tener sus aventuras amorosas extramatrimoniales y algún que otro *problemilla* con las apuestas en carreras de caballos. Las matemáticas son un bálsamo para el corazón y la mente, de eso no hay duda, pero, por suerte, no nos restan ni virtudes ni debilidades humanas.

Y ni tan mal, oye.