1. Introducción

Este artículo inaugura una serie sobre un tema apasionante: la **composición algorítmica**. Si queremos una definición concisa y breve, diríamos que la composición algorítmica se refiere al uso de algoritmos para la composición musical. En este viaje pretendemos que nuestros lectores, tanto músicos como matemáticos y en general cualquier lector curioso, comprendan los fundamentos de la teoría de algoritmos, de la composición musical y en última instancia cómo se han usado los algoritmos para componer música. En esta primera entrega trataremos los algoritmos y su definición formal y daremos ejemplos. En la segunda entrega examinaremos la definición de composición musical y sus características. En las siguientes entregas estudiaremos las principales corrientes dentro de la composición algorítmica.

2. ¿Qué es un algoritmo?

El concepto de algoritmo está asociado a la resolución de problemas. Desde este punto de vista, los algoritmos son una manera de pensar. En general, esos problemas han de ser susceptibles de ser cuantificables numéricamente y resolubles por medios matemáticos. Por ejemplo, cuando nos referimos al algoritmo de Euclides [Wik] estamos hablando de un procedimiento para resolver el problema de hallar el máximo común divisor de dos números. Sin embargo, no toda solución de un problema es un algoritmo. La solución necesita tener unas características especiales, como vamos a ver enseguida. La definición de algoritmo ha ido evolucionando según el nivel científico de la época, desde los tiempos del matemático al-Khwarizmi (siglo IX d.C.) en que algoritmo se refería a reglas aritméticas de cálculo, pasando por la formalización de Touring, hasta llegar a la definición de Donald Knuth [Knu73], una de las más aceptadas modernamente y que seguiremos aquí. Dado un problema a resolver, un **algoritmo** es un procedimiento que toma una **entrada** o valores iniciales del problema y, después realizar una serie de operaciones **bien definidas**, produce una **salida** o solución del problema. La definición de Knuth identifica cinco propiedades que un algoritmo ha de tener:

1. **Entrada:** Los valores iniciales del problema.
2. **Precisión:** Todo algoritmo tiene que estar definido de manera precisa de modo que no haya ambigüedad. En particular, los algoritmos están basados en un conjunto normalmente pequeño de operaciones básicas, que se suelen llamar **operaciones primitivas**. Estas suelen ser las operaciones matemáticas y reglas lógicas.

3. **Finitud:** Todo algoritmo tiene que terminar después de un número finito de pasos (y cuanto menor sea ese número, mejor).
4. **Salida:** Todo algoritmo ha de devolver un resultado.
5. **Efectividad:** Las operaciones que intervienen en el algoritmo han de ser suficientemente básicas.

Por supuesto, todo algoritmo que (aparentemente) resuelva un problema tiene que ir acompañado de una prueba matemática de que, en efecto, resuelve tal problema. Puesto que un algoritmo tiene que terminar en un número finito de pasos, los problemas que pueden resolverse de manera algorítmica deben tener una cierta naturaleza discreta (los problemas debe ser o bien finitos o bien si son infinitos tener una caracterización finita). Por ejemplo, el problema de enumerar todos los números primos implica dar una salida que es infinita y, en la definición dada aquí, no hay algoritmo que realice tal tarea. Sin embargo, para calcular el máximo común divisor de dos números sí es posible diseñar un algoritmo para resolver tal problema. El máximo divisor de dos números siempre existe y es un número finito comprendido entre los divisores de ambos números.

Un problema inherente a los algoritmos es su **descripción**. Los algoritmos pueden expresarse de muchas maneras: en primer lugar, en lenguaje natural, pero también como pseudocódigo y en última instancia en términos de un lenguaje de programación. Un algoritmo se puede ver como una serie de reglas formales para resolver un problema y su descripción es la enumeración de dichas reglas en el lenguaje apropiado. El inconveniente que surge al describir un algoritmo con lenguaje natural es que el grado de ambigüedad en su descripción puede ser demasiado alto porque el lenguaje natural es ambiguo. Consideremos el problema siguiente:

Problema: Dado un conjunto M de n números reales y otro número x, determinar si x está en el conjunto
M
.

Este problema es conocido como el **problema de la búsqueda**. Supongamos que los

elementos de M son $M[1], M[2], \ldots, M[n]$; note el lector que nos referimos a los elementos de M a través de un índice en notación matricial. Una manera de describir un algoritmo en lenguaje natural sería la siguiente:

Algoritmo en lenguaje natural: Para cada elemento $M[i]$ de M, con $i = 1$ hasta $i = n$, comprobar si dicho elemento es x.

Se puede apreciar que en esta descripción aparecen las características de la definición de algoritmo dadas anteriormente. Empero, esta descripción es más abstracta e ignora ciertos detalles técnicos. La idea que transmite es de que la solución se encuentra comparando cada elemento de M con x. Con frecuencia la descripción en lenguaje natural no es suficiente para detallar las ideas detrás de un algoritmo y a veces tampoco para probar su corrección. El siguiente paso es definir una serie de operaciones básicas y estructuras de datos con que describir el algoritmo. Esa descripción se llama **pseudocódigo**. Por ejemplo, el siguiente pseudocódigo corresponde al algoritmo de búsqueda.

```
BÚSQUEDA-LINEAL(M, x)

1   i ← 1
2   while i ≤ leng
3       if M [ i
4           r ← i
5           i ← length (
```

6	**if**	*i*	=	*leng*
7	**return**	*r*		

Figura 1: El algoritmo de búsqueda lineal

La entrada está especificada en la línea *BÚSQUEDA-LINEAL(M, x)* y es el conjunto *M* y el número
x
. El cuerpo del pseudocódigo contiene instrucciones de control, tal como el bucle
while
o la sentencia condicional
if
. En el pseudocódigo ya aparecen objetos matemáticos, tales como variables (la variable
i
), y operaciones entre ellos, tales como la asignación de valores, con el operador ← (líneas 4 y 5), o la comparación de valores, con el operador = (línea 6). La salida se produce en la línea 7 con la instrucción
return
. Este pseudocódigo facilita la prueba de la corrección del algoritmo. En este artículo no entraremos en la delicada cuestión de la prueba de algoritmos. Recomendamos al lector interesado acudir al magnífico libro de Cormen, Leiserson y Rivest [
CLRS01
]
Introduction to Algorithms
para profundizar en este importante tema.

En la figura 2, por último, tenemos el algoritmo codificado en lenguaje C. Como se puede observar ya no hay lenguaje natural y los detalles del algoritmo están entreverados con los detalles propios del lenguaje. Para más información sobre programación de algoritmos en lenguajes de programación, véanse [GBY91 , Sed90].

```c
#include <stdio.h>

int main()
{
   int array[100], search, c, n;

   printf("Enter the number of elements in array\n");
   scanf("%d",&n);

   printf("Enter %d integer(s)\n", n);

   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);

   printf("Enter the number to search\n");
   scanf("%d", &search);

   for (c = 0; c < n; c++)
   {
      if (array[c] == search)     /* if required element found */
      {
         printf("%d is present at location %d.\n", search, c+1);
         break;
      }
   }
   if (c == n)
      printf("%d is not present in array.\n", search);

   return 0;
}
```

[redacted] 2006.