Analizando algoritmos en búsqueda de eficiencia Durante el desarrollo de este capítulo se utilizarán dos términos claves, “ejemplar” y “problema”, que se deben diferenciar correctamente para no cometer errores. Cuando se hace referencia a un ejemplar se refiere a un caso particular de un problema (por ejemplo, determinar si en número 5 es primo). En cambio, cuando se menciona un problema hace referencia a todos los casos particulares comprendidos bajo el mismo problema (por ejemplo, determinar los números enteros primos). Un algoritmo debe funcionar correctamente en todos los ejemplares o casos del problema que resuelve. Para demostrar que un algoritmo es incorrecto basta con encontrar un ejemplar de dicho problema que no se puede resolver de manera correcta. ¿Qué significa que un algoritmo sea eficiente? Normalmente, cuando hay que resolver un problema es posible que existan distintos algoritmos adecuados para este y, obviamente, se querrá elegir el mejor. Entonces, ¿cómo elegir entre varios algoritmos para el mismo problema? Si solo hay que resolver un par de casos de un problema sencillo, seguramente no será de mayor importancia con qué algoritmo se resuelva, seguramente la elección se inclinará hacia uno que sea más rápido programar o a uno que ya esté desarrollado, sin preocuparnos por las propiedades teóricas. Pero si se tienen que resolver muchos casos o el problema es complejo, el proceso de selección del algoritmo deberá ser más cuidadoso y con criterio. Existen dos enfoques para la selección de un algoritmo: 1. Empírico (o a posteriori) consiste en programar las técnicas competidoras e ir probándolas en distintos casos en la computadora. 2. Teórico (o a priori) consiste en determinar matemáticamente la cantidad de recursos necesarios para cada uno de los algoritmos, en función del tamaño de los casos considerados. Los recursos que más nos interesan son el tiempo de computación o procesamiento (este es el más importante) y el espacio de almacenamiento, principalmente entendida como la memoria principal de la computadora. A lo largo del libro compararemos los algoritmos a través del enfoque teórico tomando como base sus tiempos de ejecución. En este sentido, se considera eficiente a un algoritmo en función a su velocidad de ejecución. [28] Hay que tener en cuenta que el tamaño de un ejemplar se corresponde formalmente con el número de bits que se requieren para representarlo en una computadora. Sin embargo, para que el análisis sea más claro y sencillo conviene ser menos formales y utilizar la palabra tamaño para indicar un entero que mida de alguna forma el número de componentes de un ejemplar. Por ejemplo cuando se habla de ordenamiento, normalmente se medirá el tamaño de un ejemplar por la cantidad de ítems que hay que ordenar, ignorando el espacio en bit que se requerirá para almacenar cada uno de estos ítems en la computadora. El enfoque teórico tiene la ventaja de que no depende ni de la computadora que se esté utilizando (hardware), ni del lenguaje de programación (software), ni siquiera de las habilidades del programador (persona). Además se ahorra el tiempo que se habría invertido en el desarrollo de un algoritmo ineficiente, como el tiempo de computación que se habría desperdiciado para comprobarlo. Y lo más importante, permite medir la eficiencia de un algoritmo para todos los casos de un problema (es decir todos los tamaños). A diferencia del teórico, el enfoque empírico muchas veces propone realizar las pruebas con ejemplar de tamaño chico y moderado, dado que el tiempo de computación requerido es elevado. Esto es una desventaja, suele suceder que los nuevos algoritmos comienzan a comportarse mejor que sus predecesores cuando aumenta el tamaño del ejemplar. Como no existe una unidad de medida para expresar la eficiencia de un algoritmo, en su lugar se la expresa según el tiempo requerido por un algoritmo. Se puede decir entonces que el algoritmo para un problema requiere un tiempo del orden de t(n) para una función dada t, si existe una constante positiva c y una implementación del algoritmo capaz de resolver todos los casos de tamaño n en un tiempo que no sea superior a ct(n) unidades de tiempo (la unidad de tiempo es arbitraria, pueden ser años, minutos, días, horas, segundos, milisegundos, etc.). Esto mismo se puede definir formalmente: T(n) = O(f(n)) si existe una constante c y un valor n0 tales que T(n)<= c f(n) cuando n>n0 Existen ciertos órdenes que se repiten con tanta frecuencia que tienen su propia denominación, como se describe a continuación, y sus funciones de crecimiento se observan en la figura 1 para valores de tamaño de uno a diez –si bien el rango de valores es pequeño– es suficiente para apreciar la diferencia entre los distintos órdenes de complejidad y cómo varía el tiempo de ejecución en base al tamaño de la entrada: 1. en el orden de O(c), o de tiempo constante; 2. en el orden de O(log n), o de tiempo logarítmico; 3. en el orden de O(n), o de tiempo lineal; 4. en el orden de O(n log n), o de tiempo casi lineal; 5. en el orden de O(n2 ), o de tiempo cuadrático; 6. en el orden de O(n3 ), o de tiempo cúbico; [29] 7. en el orden de O(nk ), o de tiempo polinómico; 8. en el orden de O(cn ), o de tiempo exponencial; 9. en el orden de O(n!), o de tiempo factorial.