y aún programas distintos que implementan el mismo algoritmo sobre la misma máquina pudieran imponer distintas restricciones. En este curso nos conformaremos con probar que nuestros algoritmos son correctos en abstracto, ignorando las limitaciones prácticas presentes en cualquier programa concreto que los implante.

2.2. La eficiencia de los algoritmos

Cuando tenemos que resolver un problema, podríamos tener disponibles varios algoritmos. Obviamente nos gustaría elegir el mejor. Esto plantea la pregunta de cómo decidir cuál de esos algoritmos es preferible. Si sólo tenemos uno o dos ejemplares por resolver, de un problema muy simple, podríamos no preocuparnos demasiado por cuál algoritmo usar: en este caso simplemente podríamos elegir al que sea más fácil de programar, o aquel para el que ya existe un programa, sin preocuparnos sobre sus propiedades teóricas. Pero si tenemos gran cantidad de ejemplares por resolver o si el problema es muy difícil de resolver, tendriamos que elegir más cuidadosamente.

El *enfoque empírico* (o *a posteriori*) para elegir un algoritmo consiste en programar los algoritmos en competencia y probarlos con ejemplares distintos y la ayuda de una computadora. El *enfoque teórico* (o *a priori*), el cual favoreceremos en este curso, consiste en determinar matemáticamente la cantidad de recursos que necesita cada algoritmo *como una funcion del tamaño de los ejemplares considerados*. Los recursos de mayor interés son el tiempo de cálculo y el espacio para almacenamiento, usualmente el primero es el más crítico, en este curso compararemos usualmente algoritmos sobre las bases de su tiempo de ejecución y cuando hablemos de la *eficiencia* de un algoritmo simplemente estaremos diciendo que tan rápido se ejecuta. Sólo ocasionalmente estaremos interesados en los requerimientos de almacenamiento de un algoritmo.

El *tamaño* de un ejemplar formalmente corresponde al número de bits necesarios para representar al ejemplar en una computadora, usando algún esquema de codificación definido precisamente y razonablemente compacto. No obstante, para ser más claro nuestro análisis, usualmente seremos menos formales y cuando usemos la palabra tamaño entenderemos cualquier entero que de alguna manera mide el número de componentes en un ejemplar. Por ejemplo, cuando hablamos sobre ordenamiento, usualmente mediremos el tamaño de un ejemplar por el número de artículos que se han de ordenar, ignorando el hecho de que cada uno de estos artículos necesitaria más de un bit para representarse sobre una computadora. Similarmente, cuando hablamos sobre grafos, usualmente mediremos el tamaño de un ejemplar por el número de nodos o vértices (o ambos) involucrados. Cuando hablemos de problemas que involucren enteros nos apartaremos de esta regla general y algunas veces daremos la eficiencia de nuestros algoritmos en términos del *valor* del ejemplar considerado y no de su tamaño (el cual sería el número de bits necesarios para representar dicho valor en binario).

La ventaja del enfoque teórico es que no depende de la computadora que se use, ni del lenguaje de computación, ni de la habilidad del programador. Ahorra, tanto el tiempo que se gastaría innecesariamente al programar un algoritmo ineficiente, como el tiempo de máquina que se gastaría probandolo. Más significativo, es que nos permite estudiar la eficiencia de un algoritmo cuando se usa sobre ejemplares de cualquier tamaño. Frecuentemente éste no es el caso con el enfoque empírico, donde consideraciones prácticas podrían forzarnos a probar nuestros algoritmos sólo sobre un número pequeño de ejemplares elegidos arbitrariamente y de tamaño moderado. Es usual que, un algoritmo descubierto recientemente, empieza a mostrar que tiene un mejor desempeño que sus predecesores sólo cuando es usado sobre ejemplares grandes, éste es un punto muy importante por el cual se prefiere el enfoque a priori.

También es posible analizar algoritmos usando un enfoque *híbrido*, donde la forma de la función que describe la eficiencia del algoritmo se determina teóricamente y luego los parámetros numéricos requeridos se determinan empíricamente para un programa y una máquina particular, al usar este enfoque podemos predecir el tiempo que una implementación real tomará para solucionar un ejemplar mucho mas grande que los usados en las pruebas. Debemos ser cuidadosos al hacer tales extrapolaciones solamente sobre las bases de un número pequeño de muestras empíricas ignorando todas las consideraciones teóricas. Las predicciones que se hacen sin soporte teórico es probable que sean muy imprecisas, sino es que totalmente equívocas.

Si queremos medir la cantidad de almacenamiento que usa un algoritmo en función de tamaño de los ejemplares, tenemos disponible una unidad natural para nosotros y se llama bit. Sin importar la máquina que se use la noción de un bit está bien definida. Si por otro lado, como en la mayoría de los casos, queremos medir la eficiencia de un algoritmo en términos del tiempo que toma para producir una respuesta, entonces no hay una elección tan obvia. Claramente no podríamos expresar esta eficiencia por ejemplo en segundos, ya que no tenemos una computadora estándar a la cual pudieran hacer referencia todas la mediciones.

Una respuesta a este problema esta dada por el *principio de invarianza*, el cual enuncia que dos implementaciones diferentes del mismo algoritmo no diferirán en eficiencia por más de alguna constante multiplicativa. Si esta constante resulta ser 5, por ejemplo, entonces sabremos que si la primera implementación toma un segundo para resolver ejemplares de cierto tamaño, entonces la segunda implementación (pudiera estar sobre una máquina distinta o escrita en un lenguaje de programación distinto) no tomará más de 5 segundos para resolver los mismos ejemplares. Más precisamente, si dos implementaciones del mismo algoritmo toman respectivamente $t_1(n)$ y $t_2(n)$ segundos para resolver un ejemplar de tamaño n, entonces siempre existen constantes positivas c y d tales que $t_1(n) \leq ct_2(n)$ y $t_2(n) \leq dt_1(n)$ siempre que n sea suficientemente grande. En otras palabras, el tiempo de ejecución de cualquiera de las dos implementaciones está acotado por un múltiplo constante del tiempo de ejecución de la otra; por lo tanto es irrelevante a cuál implementación le llamaremos primera y a cuál segunda. La condición de que n sea suficientemente grande no es realmente necesaria, como veremos posteriormente al analizar la regla del umbral. No obstante, al considerarla podemos encontrar frecuentemente constantes c y d más pequeñas. Esto es útil si tratamos de calcular buenas cotas sobre el tiempo de ejecución de una implementación cuando conocemos el tiempo de ejecución de la otra.

Este principio no es algo que podamos probar: simplemente enuncia un hecho que puede ser confirmado mediante observación. Más aún tiene una amplia aplicación. El principio se mantiene cierto sin importar cual computadora se usa para implementar un algoritmo (siempre y cuando la computadora tenga un diseño convencional), sin importar el lenguaje de programación y el compilador empleado, es más, sin importar la habilidad del programador (siempre y cuando él o ella no modifique realmente el algoritmo). Así un cambio de máquina podría permintirnos resolver un problema 10 o 100 veces más rápido lo cual nos incrementa la rapidez en un factor constante. Por otro lado, un cambio de algoritmo y sólo un cambio de algoritmo podría darnos una mejora que sería más y más marcada conforme aumenta el tamaño de los ejemplares.

Regresando a la pregunta de qué unidad será usada para expresar la eficiencia teórica de un algoritmo, el pricipio de invarianza nos permite decidir que no es necesaria tal unidad. En su lugar, sólo expresamos el tiempo que toma un algoritmo salvo una constante multiplicativa. Decimos que un algoritmo para algún problema toma un tiempo *en el orden de* $t(n)$, para una función dada t, si existe una constante positiva c y una implementación del algoritmo capaz de resolver todo ejemplar de tamaño n en no más que $ct(n)$ segundos. (Para problemas numéricos, como señalamos anteriormente, n algunas veces puede ser el valor más que el tamaño del ejemplar)

El uso de segundos en esta definición es obviamente arbitrario: sólo necesitamos cambiar la constante para acotar el tiempo por $at(n)$ años o $bt(n)$ microsegundos. Por el principio de invarianza, si cualquiera de las implementaciones del algoritmo tiene la propiedad requerida, entonces también la tendrán todos los demás, no obstante la constante multiplicativa cambiaría de una implementación a la otra. En la siguiente sección daremos un tratamiento más riguroso de este concepto importante conocido como la *notación asintótica*. A partir de la definición formal será claro porque decimos *en el orden de* en lugar del más usual *del orden de*.

Ciertos ordenes ocurren tan frecuentemente que vale la pena darles un nombre. Por ejemplo, suponga que el tiempo que toma un al-