Capítulo 1 ALGORITMIA ELEMENTAL

Nombre: Jean Carlos Iñiguez

* **Eficiencia de los algoritmos**

Al resolver un problema, a menudo hay varios algoritmos posibles. Elegir el mejor depende del contexto:

* Si se resolverán **pocos casos simples**, puede elegirse el algoritmo más fácil de programar o el ya disponible.
* Si se deben resolver **muchos casos o el problema es complejo**, hay que seleccionar el algoritmo más eficiente.

**Enfoques para comparar algoritmos:**

1. **Empírico (a posteriori)**:
   * Se implementan y prueban los algoritmos con la ayuda de una computadora.
   * Es útil, pero consume tiempo y recursos.
2. **Teórico (a priori)**:
   * Se analizan matemáticamente los recursos que requiere cada algoritmo en función del tamaño del problema.
   * Es el enfoque recomendado en el libro.

**Recursos que se analizan:**

* **Tiempo de ejecución**: el recurso más importante, es la base principal para comparar algoritmos.
* **Espacio de almacenamiento**: se considera en algunos casos específicos.
* **Otros recursos**: como número de procesadores (en algoritmos paralelos) o combinaciones como tiempo × espacio.

**Tamaño de un ejemplar:**

* Formalmente: número de bits necesarios para representarlo en una computadora.
* En la práctica: se usa una medida más intuitiva, como:
  + Número de elementos a ordenar.
  + Número de nodos o aristas en un grafo.
  + En problemas con enteros, a veces se considera directamente el valor del número (no su representación binaria).
* **Análisis de caso medio y caso peor**
* Existen algoritmos como ordenación por inserción y ordenación por selección, cuyo rendimiento depende del orden inicial de los datos.
* El caso peor ocurre cuando los datos están en el orden más desfavorable, y el caso medio es el comportamiento promedio en entradas típicas.

**Ordenación por Inserción:**

* Mejor caso (datos ya ordenados): tiempo lineal (O(n)).
* Peor caso (datos en orden inverso): tiempo cuadrático (O(n²)).
* La diferencia entre ambos puede ser muy grande (hasta 1000 veces más tiempo).

**Ordenación por Selección:**

* Tiempo de ejecución siempre cuadrático (O(n²)), sin importar el orden inicial.
* Es estable, con poca variación entre el mejor y peor caso (menos del 15%).

**Conclusión:**

* Dado que el rendimiento puede variar mucho, se analiza normalmente el caso peor para asegurar que el algoritmo funcionará bien en cualquier situación.
* **¿Qué es una operación elemental?**
* Una **operación elemental** es una acción que un algoritmo puede realizar y cuyo tiempo de ejecución se puede considerar constante. Esto significa que su duración **no depende del tamaño de los datos** que se están procesando, sino únicamente de factores externos como el tipo de computadora o el lenguaje de programación utilizado. En el análisis teórico de algoritmos, nos interesa principalmente **contar cuántas de estas operaciones se realizan**, más que el tiempo exacto que tarda cada una. Por ello, se considera que todas las operaciones elementales tienen un **coste unitario**, es decir, cada una cuenta como una unidad en el conteo total de trabajo del algoritmo.
* Entre los ejemplos más comunes de operaciones elementales están las **sumas, multiplicaciones y asignaciones**, siempre que se apliquen a datos simples (por ejemplo, números enteros pequeños). Estas operaciones suelen ejecutarse en tiempos muy cortos y predecibles, y por eso se usan como base para estimar la eficiencia de un algoritmo.
* Sin embargo, no todas las instrucciones de un programa pueden tratarse como operaciones elementales. Algunas, como el cálculo de un **factorial** o el manejo de **números muy grandes**, requieren más pasos internos y su tiempo de ejecución **sí varía según el tamaño de los datos**. Por eso, estas operaciones **no se consideran elementales**, ya que romperían la idea de “tiempo constante”.
* Un ejemplo muy ilustrativo de cómo una línea de código puede implicar muchas operaciones elementales es cuando se busca el valor mínimo de un arreglo. La instrucción x ← min{T[i] | 1 ≤ i ≤ n} parece sencilla, pero internamente se está comparando cada elemento con el actual mínimo, lo que implica **n – 1 comparaciones**, es decir, **n – 1 operaciones elementales**.
* En conclusión, el concepto de operación elemental es fundamental en el análisis de algoritmos, ya que nos permite estimar la eficiencia con independencia de los detalles técnicos de implementación. Al **simplificar y asumir un coste unitario para estas operaciones básicas**, podemos enfocarnos en el comportamiento general del algoritmo conforme crece el tamaño del problema.
* **¿Por qué hay que buscar la eficiencia?**

A pesar de que las computadoras se vuelven más rápidas con el tiempo, **la eficiencia de los algoritmos sigue siendo fundamental**. Confiar únicamente en el hardware no es suficiente cuando se enfrentan problemas cuya solución requiere un gran número de pasos.

**Por ejemplo:**

* Si usamos un **algoritmo exponencial**, el tiempo de ejecución crece extremadamente rápido.  
  Un problema de tamaño n=10n = 10n=10 puede resolverse en décimas de segundo, pero al aumentar a n=20n = 20n=20, el tiempo se multiplica mil veces.  
  Y para n=30n = 30n=30, el tiempo requerido puede superar un día.
* Incluso comprando una **computadora 100 veces más rápida**, la mejora es mínima.  
  Solo permite resolver problemas de tamaño n+7n + 7n+7 en el mismo tiempo, debido al crecimiento exponencial del algoritmo.
* En cambio, si se mejora el algoritmo —por ejemplo, reemplazando uno exponencial por uno **cúbico (O(n3)O(n^3)O(n3))**— los beneficios son enormes.  
  El mismo problema que antes tardaba días en resolverse, ahora se puede resolver en minutos, y se pueden manejar casos de tamaño mucho mayor (hasta n=1500n = 1500n=1500 en un año de cálculo).

**Este análisis deja claro que:**

* **La eficiencia del algoritmo tiene mucho más impacto** que la velocidad del hardware.
* **Invertir en mejores algoritmos es una mejor estrategia** que invertir solo en máquinas más rápidas.
* La eficiencia permite **resolver problemas reales a gran escala**, algo que el hardware por sí solo no puede lograr.