* 1. **Tiempo de ejecución**
     1. **Concepto**

El tiempo de ejecución se refiere a la cantidad de tiempo que tarda un algoritmo en completar su tarea para una entrada de datos dada. Es una métrica fundamental para evaluar la eficiencia de un algoritmo y determinar su escalabilidad a medida que aumenta el tamaño de los datos de entrada.

* + 1. **Notación**

Existen diferentes notaciones para expresar el tiempo de ejecución de un algoritmo, siendo la notación asintótica la más común. La notación asintótica se enfoca en el comportamiento del algoritmo cuando el tamaño de la entrada crece indefinidamente, ignorando factores constantes y términos menos significativos. Las notaciones más utilizadas son:

* **Notación O grande (O):** Representa el peor caso o el límite superior del tiempo de ejecución del algoritmo.
* **Notación Omega (Ω):** Representa el mejor caso o el límite inferior del tiempo de ejecución del algoritmo.
* **Notación Theta (Θ):** Representa el caso promedio o el orden exacto del tiempo de ejecución del algoritmo.
  + 1. **Ejemplo**

Supongamos que tenemos un algoritmo de búsqueda lineal en un arreglo de n elementos. En el mejor caso, el elemento buscado se encuentra en la primera posición, por lo que el tiempo de ejecución es Ω(1). En el peor caso, el elemento buscado se encuentra en la última posición o no está presente, por lo que el tiempo de ejecución es O(n). En el caso promedio, el tiempo de ejecución es Θ(n).

1. **Aporte matemático**
   * 1. **Complejidad temporal**

La complejidad temporal se refiere al tiempo de ejecución de un algoritmo en función del tamaño de la entrada. Proporciona una medida cuantitativa de la eficiencia del algoritmo

El análisis de la complejidad temporal se realiza considerando tres casos principales: el mejor caso, el caso promedio y el peor caso. Esto permite comprender el comportamiento del algoritmo en diferentes escenarios.

Se utilizan herramientas matemáticas, como el álgebra y el cálculo, para determinar la complejidad temporal de un algoritmo. Esto implica analizar el número de operaciones realizadas en función del tamaño de la entrada y expresar la complejidad mediante notaciones asintóticas.

* + 1. **Algoritmos**

Los algoritmos pueden ser representados de manera formal utilizando notaciones matemáticas, como pseudocódigo, diagramas de flujo o descripciones matemáticas precisas.

Se utilizan técnicas de demostración matemática, como la inducción, para probar la corrección de un algoritmo, asegurando que produzca resultados correctos para todas las entradas válidas. Además, se demuestra la terminación del algoritmo, garantizando que finalice en un tiempo finito.

Muchos algoritmos recursivos pueden ser analizados mediante relaciones de recurrencia, que describen la relación entre el costo de una instancia del problema y el costo de sus subproblemas. Se utilizan métodos matemáticos, como el método de sustitución, el método de iteración y el método del árbol de recursión, para resolver estas relaciones de recurrencia.

* + 1. **Notación Big O**

La notación Big O se utiliza para describir el comportamiento asintótico de la complejidad temporal de un algoritmo a medida que el tamaño de la entrada crece. Proporciona una cota superior en el peor caso**.**

Existen reglas y propiedades matemáticas que rigen el cálculo y la manipulación de la notación Big O, como la aditividad, la multiplicación y la transitividad.

La notación Big O permite comparar la eficiencia de diferentes algoritmos que resuelven el mismo problema, al observar cómo se comporta su complejidad temporal a medida que aumenta el tamaño de la entrada.

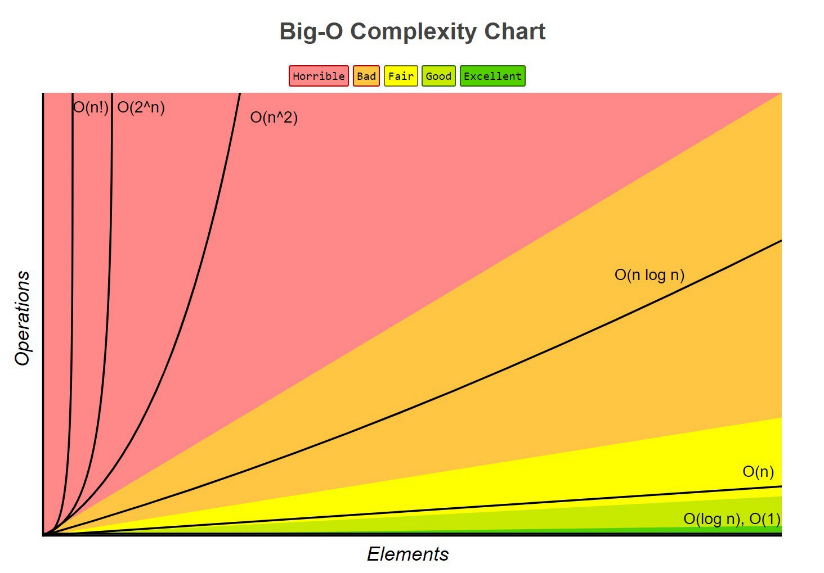
**Figura 1**

Figura 1: Tabla de notación Big O

* 1. **Ejemplo en código**

Main.cpp  
**int** **main**() {

Validaciones validador;

ListaSimple lista;

**int** cantidad;

cout << " \*\*\*\* TIEMPO DE EJECUCION DE UN ALGORITMO \*\*\*\*\* **\n**"<<endl;

cout << " \*\* Ordenamiento Burbuja \*\* **\n**"<<endl;

cout << "- Ingresa la cantidad de elementos en la lista: ";

cantidad = validador.validarEnteros();

cout << "**\n**";

**if** (cantidad <= **0**) {

cout << "La cantidad de elementos debe ser mayor que 0." << endl;

**return** **1**;

}

cout << ">> Ingresa " << cantidad << " numeros:" << endl;

**for** (**int** i = **0**; i < cantidad; i++) {

**int** valor;

valor = validador.validarEnteros();

cout << "**\n**";

lista.agregarElemento(valor);

}

**auto** inicio = std::chrono::high\_resolution\_clock::now();

cout << "**\n**Lista original: ";

lista.mostrarLista();

lista.ordenarBurbuja();

**auto** fin = std::chrono::high\_resolution\_clock::now();

cout << "Lista ordenada: ";

lista.mostrarLista();

std::chrono::duration<**double**> tiempo = fin - inicio;

std::cout << "**\n\n**====================================================================**\n\n**";

std::cout << "- TIEMPO DE EJECUCION DEL ALGORTIMO: " << std::endl;

std::cout << " " << tiempo.count() << " segundos**\n**" << std::endl;

std::cout << "======================================================================**\n\n**";

**return** **0**;

}

**Ejecución**

**Figura 2**

**Texto

Descripción generada automáticamente**

Figura 2: Ejecución de código

1. **Recomendaciones**

Analizar cuidadosamente el tiempo de ejecución de los algoritmos propuestos utilizando la notación asintótica y compararlo con soluciones alternativas.

Priorizar algoritmos con mejores complejidades de tiempo, especialmente aquellos que son O(log n) o O(n log n), ya que tienden a ser más escalables que algoritmos cuadráticos o exponenciales.

Optimizar el código implementado para minimizar operaciones innecesarias y aprovechar técnicas como memorización, división y conquista, o programación dinámica cuando sea posible.

1. **Conclusiones**

El análisis del tiempo de ejecución es fundamental para evaluar la eficiencia de un algoritmo. Nos permite comprender cómo se comporta un algoritmo a medida que crece el tamaño de la entrada de datos, lo que es crucial para determinar su escalabilidad y viabilidad en situaciones del mundo real.

La notación big-O, en particular, es ampliamente utilizada para expresar el tiempo de ejecución en el peor caso, lo que nos brinda una cota superior del rendimiento del algoritmo. Esto es crucial para garantizar que un algoritmo pueda manejar las entradas más desafiantes dentro de un tiempo razonable.

Al comparar el tiempo de ejecución de diferentes algoritmos, podemos identificar aquellos que son más eficientes y escalables, lo que nos permite tomar decisiones informadas al seleccionar el enfoque más apropiado para un problema dado.

**Referencias**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.

Sedgewick, R., & Wayne, K. (2021). Algorithms (4th ed.). Addison-Wesley Professional.

Knuth, D. E. (1997). The art of computer programming, volume 1: Fundamental algorithms (3rd ed.). Addison-Wesley.