**Universidad Andrés Bello**
Facultad de Ingeniería

Universidad Andrés Bello®

Tarea 4 - Análisis y Diseño de Algoritmos

Cristóbal Flores Villegas

21.324.739-5

Nrc: 8218

**Universidad Andrés Bello**
Facultad de Ingeniería

# 1. Introducción

En esta tarea se busca **examinar empíricamente** el tiempo de ejecución que toma multiplicar matrices cuadradas de distintos tamaños, y **contrastar** estos resultados con la **complejidad teórica** del algoritmo convencional de multiplicación de matrices. Concretamente, se realizan los siguientes pasos:

1. Se crea una función para multiplicar dos matrices de tamaño ***n×n*** y medir el tiempo que demora.
2. Se generan matrices aleatorias de diferentes tamaños en el rango [2, 10.000].
3. Se grafica el tiempo de ejecución contra el tamaño n.
4. Se discute la semejanza entre los resultados experimentales y la complejidad esperada.

---

# 2. Metodología

- **Generación de matrices**
  Se hace uso de la función `np.random.rand(n, n)` para construir matrices aleatorias de tamaño n×n. Esta aproximación simplifica la experimentación al no requerir datos específicos.
- **Multiplicación y medición de tiempo**
  La multiplicación se ejecuta con `np.dot(A, B)`. Se calcula el tiempo mediante `time.time()` antes y después de la operación. Esto se encapsula en una función que retorna tanto el resultado como el tiempo de ejecución:

  ```
  python
  CopiarEditar
  def multiply_matrices(A, B):
      start_time = time.time()
      C = np.dot(A, B)
      end_time = time.time()
      return C, (end_time - start_time)
  ```

- **Configuración del experimento**
  - Se define un rango de tamaños n desde 2 hasta 10.000, distribuidos en 500 puntos.
  - Para cada n, se generan dos matrices AAA y BBB y se registra el tiempo de ejecución.
  - Finalmente, se grafica la relación n vs. tiempo de ejecución con la librería `matplotlib`.

---

# 3. Análisis de Resultados

En el archivo multiplicacion_matrices.png, se aprecia un incremento **marcado** del tiempo de ejecución al crecer n. Este comportamiento se alinea con la **complejidad de orden O(n3)**, característica de la multiplicación de matrices por el método estándar. La justificación se basa en que para cada entrada de la matriz resultado (n^2 en total) se efectúan n productos y sumas, resultando aproximadamente en n^3 operaciones.

Si bien numpy.dot puede incorporar optimizaciones internas, la dependencia principal se mantiene cercana a la de una función cúbica. El aumento de tiempo observado en el gráfico refuerza este hecho: a mayor n, el costo crece de manera acorde con O(n3).

---

# 4. Conclusiones

1. **Relación teórica y práctica**: Los resultados experimentales confirman la **complejidad cúbica** para la multiplicación de matrices en su versión clásica.
2. **Escalabilidad**: Para valores de n muy grandes, el tiempo de ejecución se eleva rápidamente, lo que subraya la necesidad de algoritmos más sofisticados o hardware especial en aplicaciones de gran escala.
3. **Relevancia**: Este análisis ilustra la importancia de entender cómo crece el tiempo de ejecución al aumentar el tamaño de la entrada, un aspecto fundamental en el **diseño y evaluación de algoritmos**.