TAREA 1– 2025-2
ALGORITMOS Y COMPLEJIDAD

«Análisis experimental de la complejidad de algoritmos de ordenamiento y multiplicación de matrices.»

Fecha publicación: **11 agosto de 2025**

Fecha límite de entrega: **08 de septiembre de 2025**

Equipo ALGORITMOS Y COMPLEJIDAD 2025-2

Índice

1. Objetivos

El objetivo de esta tarea es introducirnos en el Análisis experimental de Algoritmos. Para ello, se propone realizar un estudio experimental de 4 algoritmos de ordenamiento y 2 de multiplicación de matrices.

Para cada uno de los dos problemas, el de ordenar un arreglo unidimensional de números enteros, y el de multiplicar dos matrices cuadradas, existen múltiples algoritmos que los resuelven, y la mayoría, si no todos, tienen implementaciones de fácil acceso (ya se encuentran en bibliotecas de lenguajes de programación, o son fáciles de encontrar en Internet). Sus complejidades teóricas son conocidas, ya que estás son una característica fundamental a la hora de diseñar algoritmos y, en la mayoría de los casos, la motivación para diseñarlos.

El desafío en esta tarea no está en diseñar los algoritmos (los encuentra fácilmente), ni tampoco en calcular o demostrar su complejidad teórica: el desafío está en **estudiar su comportamiento en la práctica** y relacionar este comportamiento con la complejidad teórica.

2. Especificaciones

En esta sección se describen los pasos a seguir para completar la tarea, la cual consiste en desarrollar código en C++ «Implementaciones» y en realizar un informe «Informe». Se espera que cada uno de los pasos se realice de manera ordenada y siguiendo las instrucciones dadas.

Abra este documento en algún lector de pdf que permita hipervínculos, ya que en este documento el texto en color azul suele indicar un hipervínculo.

(1) En caso de dudas, enviar preguntas **al foro de la Tarea 1**.

En caso de cualquier de modificaciones, todas se informarán tanto por aula como por discord.

(2) Todo lo necesario para realizar la tarea se encuentra en el repositorio de github que se obtiene mediante la siguiente invitación de GitHub Classroom:

`https://classroom.github.com/a/0NcKwA6A`

Nota: La estructura de archivos y el template del proyecto se encuentran disponibles en el repositorio generado.

(3) El repositorio generado contiene toda la información oficial y estructura necesaria para completar la tarea.

2.1. Implementaciones

(1) **Ordenamiento y Multiplicación.**

- **Ordenamiento.** Implementar en C++ los algoritmos de ordenamiento: INSERTION SORT, MERGE SORT, QUICK SORT, PANDA SORT y el algoritmo SORT de la librería estándar de C++. Use la siguiente estructura de archivos:
  - code/sorting/algorithms/insertionsort.cpp
  - code/sorting/algorithms/mergesort.cpp
  - code/sorting/algorithms/quicksort.cpp
  - code/sorting/algorithms/pandasort.cpp
  - code/sorting/algorithms/sort.cpp

  Importante para Quick Sort: NO implementar el QuickSort básico. Debe implementar una variación del algoritmo, eligiendo una de las siguientes opciones:
  - **Random Pivot** - Selección aleatoria del pivot
  - **Middle Element** - La mediana siempre es el pivot
  - **Median-of-Medians** - Algoritmo determinístico O(n) para pivot óptimo
  - **Randomized QuickSort** - Con shuffle inicial del arreglo

  Puede utilizar implementaciones disponibles en internet, pero debe **citar debidamente las fuentes** en el código y en el informe.

  Nota importante: El algoritmo PANDA SORT es un algoritmo de ordenamiento desarrollado específicamente para esta tarea. La implementación completa se encuentra disponible en el repositorio del curso para su análisis.

  El algoritmo SORT ya se encuentra implementado en el repositorio con el fin de que sea incluido en sus mediciones del punto (2) de la subsección «Implementaciones».
- **Multiplicación.** Se deben implementar en C++ los algoritmos de multiplicación NAIVE y STRASSEN. Use la siguiente estructura de archivos.
  - code/matrix_multiplication/algorithms/naive.cpp
  - code/matrix_multiplication/algorithms/strassen.cpp

(2) **Mediciones de tiempo y memoria.** Implementar el programa que realizará las mediciones de tiempo y memoria en C++ (programas principales) y su respectivo `makefile`, que ejecutará los algoritmos y generará los archivos de salida en cada uno de los directorios measurements sorting y measurements matrix multiplication con los resultados de cada uno de los algoritmos.

- code/sorting/sorting.cpp
- code/matrix_multiplication/matrix_multiplication.cpp

(3) **Generación de gráficos.** Implementar el programa que generará los gráficos en PYTHON y que se encargará de leer los archivos generados por los programas principales guardados en measurements sorting y measurements matrix multiplication, para luego graficar los resultados obtenidos y guardarlos en formato PNG en plots sorting y plots matrix multiplication.

- code/sorting/scripts/plot_generator.py
- code/matrix_multiplication/scripts/plot_generator.py

(4) **Documentación.** Documentar cada uno de los pasos anteriores

- Completar el archivo `README.md` del directorio `code`
- Documentar en cada uno de sus programas, al inicio de cada archivo, **fuentes de información, referencias y/o bibliografía** utilizada para la implementación de cada uno de los algoritmos.
- Para el algoritmo PANDA SORT, documentar el diseño y la lógica del algoritmo desarrollado.

2.2. Informe

Generar un informe en LaTeX que contenga los resultados obtenidos y una discusión sobre ellos. En el repositorio generado por GitHub Classroom podrá encontrar el Template en el directorio report que **deben utilizar**, en esta entrega:

Template disponible en: `report/` dentro del repositorio generado

- No se debe modificar la estructura del informe.
- Las indicaciones se encuentran en el archivo `README.md` del repositorio y en la plantilla de LaTeX.

3. Condiciones de entrega

(1) La tarea es **individual**, sin excepciones.

(2) La entrega debe realizarse vía aula.usm.cl, entregando la url del repositorio de GitHub donde se encuentra el código fuente de su tarea.

Debe aceptar la invitación de GitHub Classroom para obtener su repositorio de trabajo:

https://classroom.github.com/a/0NcKwA6A

- El repositorio generado por GitHub Classroom será automáticamente privado y contendrá toda la estructura y template necesarios para completar la tarea.

(3) Si se utiliza algún código, idea, o contenido extraído de otra fuente, este **debe** ser citado en el lugar exacto donde se utilice.

(4) Asegúrese que todas sus entregas tengan sus datos completos: número de la tarea, ramo, semestre, nombre y rol. Puede incluirlas como comentarios en sus fuentes LaTeX (en TeX comentarios son desde % hasta el final de la línea) o en posibles programas. Anótese como autor de los textos.

(5) Si usa material adicional al discutido en clases, detállelo. Agregue información suficiente para ubicar ese material (en caso de no tratarse de discusiones con compañeros de curso u otras personas).

(6) No modifique `preamble.tex`, `report.tex`, `rules.tex`, estructura de directorios, nombres de archivos, configuración del documento, etc. Sólo agregue texto, imágenes, tablas, código, etc. En el códigos fuente de su informe/reporte, no agregue paquetes, ni archivos .tex.

(7) La fecha límite de entrega es el día **08 de septiembre de 2025** .

NO SE ACEPTARÁN TAREAS FUERA DE PLAZO.

(8) Nos reservamos el derecho de llamar a interrogación sobre algunas de las tareas entregadas. En tal caso, la nota de la tarea será la obtenida en la interrogación.

NO PRESENTARSE A UN LLAMADO A INTERROGACIÓN SIN JUSTIFICACIÓN PREVIA SIGNIFICA AUTOMÁTICAMENTE NOTA 0.

A. Casos de prueba

A.1. Ordenamiento de un arreglo unidimensional de números enteros

Entrada:

- Leer un arreglo unidimensional A desde el archivo `{n}_{t}_{d}_{m}.txt`.
  - n hace referencia a la cantidad de elementos (o largo del arreglo) y pertenece al conjunto $\mathcal{N} = \{10^1, 10^3, 10^5, 10^7\}$.
  - t hace referencia al tipo de matriz, y pertenece al conjunto $\mathcal{T} = \{\text{ascendente}, \text{descendente}, \text{aleatorio}\}$.
  - d hace referencia al conjunto dominio de cada elemento del arreglo. $d = \{D1, D7\}$, donde $D7$ implica que el dominio es $\{0,1,2,3,4,5,6,7,8,9\}$ y $D7$ que el dominio es $\{0,1,2,3,...,10^7\}$.
  - m hace referencia a la muestra aleatoria (o caso de prueba) y pertenece al conjunto $\mathcal{M} = \{a, b, c\}$.

Salida:

- Escribir la matriz resultante $M_1 \times M_2$ en el archivo `{n}_{t}_{d}_{m}_out.txt`.

Los programas que generan estos arreglos se encuentran en

code/sorting/scripts/array_generator.py

A.2. Multiplicación de matrices cuadradas de números eneteros

Entrada:

- Leer dos matrices cuadradas de entrada M_1 y M_2 desde los archivos `{n}_{t}_{d}_{m}_1.txt` y `{n}_{t}_{d}_{m}_2.txt`, respectivamente.
  - n hace referencia a la dimensión de la matriz (n filas y n columnas) y pertenece al conjunto $\mathcal{N} = \{2^4, 2^6, 2^8, 2^{10}\}$.
  - t hace referencia al tipo de matriz, y pertenece al conjunto $\mathcal{T} = \{\text{dispersa}, \text{diagonal}, \text{densa}\}$.
  - d hace referencia al dominio de cada coeficiente de la matriz $d = \{D0, D10\}$, donde $D0$ implica que el dominio es $\{0,1\}$ y $D10$ que el dominio es $\{0,1,2,3,4,5,6,7,8,9\}$.
  - m hace referencia a la muestra aleatoria (o caso de prueba) y pertenece al conjunto $\mathcal{M} = \{a, b, c\}$.

Salida:

- Escribir la matriz resultante $M_1 \times M_2$ en el archivo `{n}_{t}_{d}_{m}_out.txt`.

Los programas que generan estas matrices se encuentran en

code/matrix_multiplication/scripts/matrix_generator.py