Trabajo Práctico: Algoritmos de búsqueda y ordenamiento

Estructuras de datos
Comision A (Turno Mañana)
Segundo cuatrimestre de 2020

Docente:	Ariel Clocchiatti
Fecha de entrega:	21 / 09 / 2020
Email:	aclocchi@gmail.com

Índice

1. Introducción

En este trabajo práctico vamos a trabajar con la integración de los temas que vimos hasta el momento en la cursada para abordar los problemas de búsqueda y ordenamiento de los elementos de un arreglo:

- Programación en Python.
- Algoritmos recursivos.
- Arreglos.

1.1. Algoritmos de búsqueda y ordenamiento

Los algoritmos de búsqueda y ordenamiento son muy importantes, ya que son ampliamente utilizados en la resolución de problemas informáticos. En la mayoría de los casos, trabajar con datos ordenados hace que los algoritmos sean más eficientes. La búsqueda es uno de los problemas que más se beneficia al trabajar con datos ordenados. Por lo cual, estos algoritmos están muy relacionados entre sí.

Para simplificar vamos a asumir que los arreglos son unidimensionales y contienen solo números enteros (aunque obviamente es posible usar cualquier estructura de datos). También vamos a suponer que el algoritmo completo se puede resolver en memoria, osea que vamos a trabajar con arreglos de un tamaño reducido y los ordenamientos serán de tipo numérico y de menor a mayor.

1.2. Eficiencia y complejidad

Un factor clave en los algoritmos de búsqueda y ordenamiento es la eficiencia. Cuanto menor sea la complejidad computacional, los algoritmos son más eficientes y por consiguiente, más rápidos. La complejidad depende de la cantidad de datos que tenemos que procesar. Cuando tenemos un algoritmo para resolver un problema, tenemos que medir los recursos computacionales que el algoritmo va a necesitar:

- Tiempo de ejecución en función de la cantidad de datos (cantidad de operaciones que realiza el algoritmo).
- Cantidad de memoria en función de la cantidad de datos.

Estas dos medidas conforman la complejidad computacional del algoritmo. Como ya aclaramos antes, para simplificar suponemos que los algoritmos que vamos a estudiar, se resuelven completos en memoria, osea que no tenemos problema con la complejidad espacial. Así que vamos a ver la idea de complejidad temporal.

La complejidad temporal mide la cantidad de operaciones básicas que realiza el algoritmo (sumas, restas, multiplicaciones, divisiones, operadores de comparación, operadores lógicos, etc), y se representa usando una notación que quiere decir "del orden de..."($O(...)$).

Veamos algunos ejemplos de complejidades y que quieren decir, en todos los casos n es la cantidad de datos, por ejemplo, el tamaño del arreglo que queremos ordenar o en el que queremos buscar:

- $O(log_2(n))$: Complejidad logarítmica, es decir, la cantidad de operaciones que hace el algoritmo esta en el orden del log_2 de la cantidad de datos.
- $O(n)$: Complejidad de orden lineal, es decir, la cantidad de operaciones que hace el algoritmo esta en el orden de la cantidad de datos.
- $O(nlog_2(n))$: Complejidad lineal por logarítmica.
- $O(n^2)$: Complejidad cuadrática.

- $O(2^n)$: Complejidad exponencial.

En el gráfico de abajo pueden ver como son estas complejidades en función de la cantidad de datos, para ver cual es mejor, recuerden que a menor complejidad, más eficiencia.

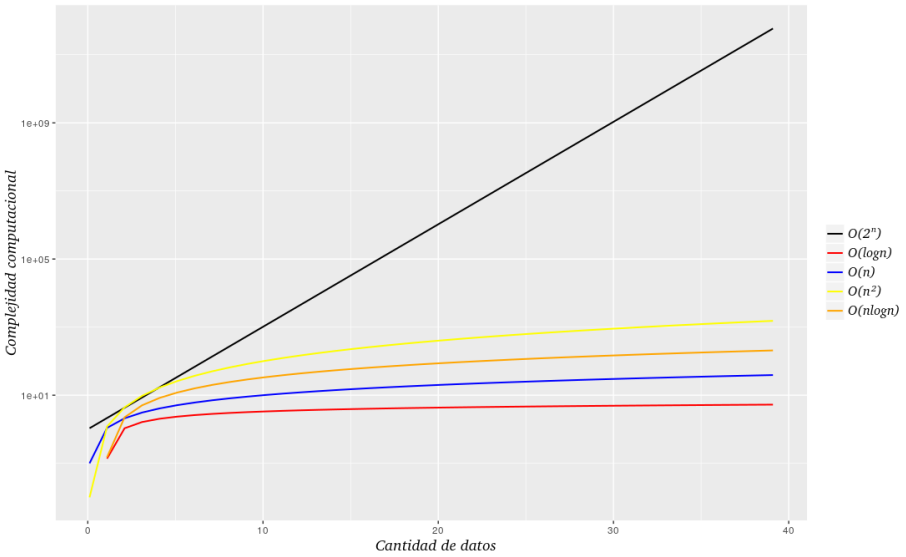

Figura 1: Complejidad en función de la cantidad de datos (Escala logarítmica).

Cuando vean la complejidad computacional de los algoritmos que estamos estudiando, van a ver que se habla de:

- Mejor caso.
- Promedio.
- Peor caso.

Esto pasa porque los algoritmos no funcionan igual para cualquier conjunto de datos de entrada, entonces hay casos donde funcionan peor que con otros. Cuando comparen las complejidades de los algoritmos en este trabajo, usen los valores de "Promedio".

2. Objetivos

Los objetivos de este trabajo práctico son:

- Comprender los conceptos de algoritmos de búsqueda y ordenamiento.
- Investigar y estudiar el funcionamiento de los algoritmos.
- Implementar los algoritmos en Python.
- Comparar la eficiencia de los distintos algoritmos según su complejidad computacional.

Vamos a ver algunos de los algoritmos de búsqueda y ordenamiento que existen. Ustedes tienen que buscar información sobre el funcionamiento y la complejidad computacional de cada uno de ellos (No tienen que calcularla ustedes la complejidad, solo buscarla y poder comparar la eficiencia de los algoritmos entre sí). Luego deben estudiar su funcionamiento y finalmente implementar alguno de los algoritmos según el que le toque a cada grupo.

Los algoritmos con los que vamos a trabajar son:

- Búsqueda
  - Búsqueda lineal o secuencial (Linear search).
  - Búsqueda binaria (Binary search).
- Ordenamiento
  - Ordenamiento por inserción (Insertion sort).
  - Ordenamiento por selección (Selection sort).
  - Ordenamiento de burbuja (Bubble sort).
  - Ordenamiento por mezcla (Merge sort).
  - Ordenamiento rápido (Quicksort).

3. Asignación de algoritmos por grupos

Cada grupo debe implementar los algoritmos que le toquen, según la siguiente tabla:

Nro de grupo	Algoritmo	
	Búsqueda	Ordenamiento
1	Búsqueda binaria	Ordenamiento rápido
2	Búsqueda binaria	Ordenamiento de burbuja
3	Búsqueda binaria	Ordenamiento por mezcla
4	Búsqueda binaria	Ordenamiento por inserción
5	Búsqueda binaria	Ordenamiento por selección
6	Búsqueda binaria	Ordenamiento rápido

Tabla 1: Algoritmos a implementar por grupo

Todas las implementaciones tienen que ser recursivas. Por su puesto, aconsejamos que modularicen el código y usen funciones y/o procedimientos accesorios. No hace falta que las funciones y/o procedimientos accesorios sean recursivos, la que si o si tiene que ser recursiva es la función que hace la búsqueda o el ordenamiento. Por ejemplo, si para implementar el ordenamiento de burbuja, usan una función accesoria que calcula el máximo entre dos números, esta no tiene que ser recursiva. La función que hace el ordenamiento si debe serlo.

4. Informe

La entrega del trabajo práctico debe ser con un informe escrito (doc, pdf), que incluya:

- Descripción escrita del funcionamiento de los algoritmos estudiados e implementados (los que le toca a cada grupo). Pueden incluir gráficos y diagramas de flujo.
- Comparación de los algoritmos estudiados con los otros algoritmos del mismo tipo, teniendo en cuenta la complejidad computacional. Por ejemplo, si a su grupo les toca el algoritmo de ordenamiento rápido, deben incluir la comparación de ese algoritmo con los otros 4 algoritmos de ordenamiento propuestos. Lo mismo con los de búsqueda.
- Código completo y comentado de la implementación.
- Opcional: video explicando cómo funciona el algoritmo.

Luego de la entrega, les vamos a hacer preguntas a las/los integrantes del grupo sobre el trabajo, así que les aconsejamos no copiar código que encuentren en internet. Aclaración: Se deben usar arreglos de los que vimos en la cursada, **no se pueden usar listas**.

5. Bibliografía

Algunos PDFs y cosas que pueden leer, por supuesto les recomendamos que busquen en internet, ya que hay mucho material sobre estos algoritmos:

- http://novella.mhhe.com/sites/dl/free/844814077x/619434/A06.pdf
- https://runestone.academy/runestone/static/pythoned/SortSearch/toctree.html (Contiene animaciones de los algoritmos funcionando)