Algoritmos y Estructuras de Datos

Guía Práctica 8
Sorting
Primer Cuatrimestre 2025

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. Comparar la complejidad de los algoritmos de ordenamiento dados en la teórica para el caso en que el arreglo a ordenar se encuentre perfectamente ordenado de manera *inversa* a la deseada.

Ejercicio 2. Defina la propiedad de estabilidad en un algoritmo de ordenamiento. Explique por qué qué el algoritmo de *heapSort* no es estable.

Ejercicio 3. Imagine secuencias de naturales de la forma $s = Concatenar(s', s'')$, donde s' es una secuencia ordenada de naturales y s'' es una secuencia de naturales elegidos al azar. ¿Qué método utilizaría para ordenar s? Justificar. (Entiéndase que s' se encuentra ordenada de la manera deseada.)

Ejercicio 4. Escribir un algoritmo que encuentre los k elementos más chicos de un arreglo de dimensión n, donde $k \leq n$. ¿Cuál es su complejidad temporal? ¿A partir de qué valor de k es ventajoso ordenar el arreglo entero primero?

Ejercicio 5. Se tiene un conjunto de n secuencias $\{s_1, s_2, \ldots, s_n\}$ en donde cada s_i $(1 \leq i \leq n)$ es una secuencia ordenada de naturales. ¿Qué método utilizaría para obtener un arreglo que contenga todos los elementos de la unión de los s_i ordenados. Describirlo. Justificar.

Ejercicio 6. Se tiene un arreglo de n números naturales que se quiere ordenar por frecuencia, y en caso de igual frecuencia, por su valor. Por ejemplo, a partir del arreglo [1, 3, 1, 7, 2, 7, 1, 7, 3] se quiere obtener [1, 1, 1, 7, 7, 7, 3, 3, 2]. Describa un algoritmo que realice el ordenamiento descripto, utilizando las estructuras de datos intermedias que considere necesarias. Calcule el orden de complejidad temporal del algoritmo propuesto.

Ejercicio 7. Sea $A[1 \ldots n]$ un arreglo que contiene n números naturales. Diremos que un rango de posiciones $[i \ldots j]$, con $1 \leq i \leq j \leq n$, contiene una escalera en A si valen las siguientes dos propiedades:

1. $(\forall k : \mathbb{N})(i \leq k < j) \implies (A[k+1] = A[k] + 1)$ (esto es, los elementos no sólo están ordenados en forma creciente, sino que además el siguiente vale exactamente uno más que el anterior).
2. Si $1 < i$ entonces $A[i] \neq A[i-1] + 1$ y si $j < n$ entonces $A[j+1] \neq A[j] + 1$ (la propiedad es *maximal*, es decir que el rango no puede extenderse sin que deje de ser una escalera según el punto anterior).

Se puede verificar fácilmente que cualquier arreglo puede ser descompuesto de manera única como una secuencia de escaleras. Se pide escribir un algoritmo para reposicionar las escaleras del arreglo original, de modo que las mismas se presenten en orden decreciente de longitud y, para las de la misma longitud, se presenten ordenadas en forma creciente por el primer valor de la escalera.

El resultado debe ser del mismo tipo de datos que el arreglo original. Calcule la complejidad temporal de la solución propuesta, y justifique dicho cálculo.

Por ejemplo, el siguiente arreglo

5	6	8	9	10	7	8	9	20	15

debería ser transformado a

7	8	9	8	9	10	5	6	15	20

Ayuda: se aconseja comenzar el ejercicio con una clara descripción en castellano de la estrategia que propone para resolver el problema.

Ejercicio 8. Suponga que su objetivo es ordenar arreglos de naturales (sobre los que no se conoce nada en especial), y que cuenta con una implementación de árboles AVL. ¿Se le ocurre alguna forma de aprovecharla? Conciba un algoritmo que llamaremos *AVL Sort*. ¿Cuál es el mejor orden de complejidad que puede lograr?

Ayuda: debería hallar un algoritmo que requiera tiempo $O(n \log d)$ en peor caso, donde n es la cantidad de elementos a ordenar y d es la cantidad de elementos distintos. Para inspirarse, piense en *Heap Sort* (no en los detalles puntuales, sino en la idea general de este último).

Justifique por qué su algoritmo cumple con lo pedido.

Ejercicio 9. Se tienen dos arreglos de números naturales, $A[1..n]$ y $B[1..m]$. Nada en especial se sabe de B, pero A tiene n' secuencias de números repetidos continuos (por ejemplo $A = [333331111888887777145555], n' = 7$). Se sabe además que n' es mucho más chico que n. Se desea obtener un arreglo C de tamaño $n+m$ que contenga los elementos de A y B, ordenados.

1. Escriba un algoritmo para obtener C que tenga complejidad temporal $O(n + (n' + m)\log(n' + m))$ en el peor caso. Justifique la complejidad de su algoritmo.
2. Suponiendo que todos los elementos de B se encuentran en A, escriba un algoritmo para obtener C que tenga complejidad temporal $O(n + n'(\log(n') + m))$ en el peor caso y que utilice solamente arreglos como estructuras auxiliares. Justifique la complejidad de su algoritmo.

Ejercicio 10. Considere la siguiente estructura para guardar las notas de un alumno de un curso:

`alumno` es `struct`⟨nombre: `string`, *turno*: `Turno`, *puntaje*: $\mathbb{N}$⟩
donde `Turno` es `enum{Mañana, Noche}` y *puntaje* es un $\mathbb{N}$ no mayor que 10.

Se necesita ordenar un arreglo de alumnos de forma tal que todes les alumnes de la mañana aparezcan al inicio de la tabla según un orden creciente de notas y todos los de la noche aparezcan al final de la tabla también ordenados de manera creciente respecto de su puntaje, como muestra en el siguiente ejemplo:

Entrada

Ana	*Mañana*	10
Rita	*Noche*	6
Juan	*Mañana*	6
Paula	*Mañana*	7
Jose	*Noche*	7
Pedro	*Noche*	8

Salida

Juan	*Mañana*	6
Paula	*Mañana*	7
Ana	*Mañana*	10
Rita	*Noche*	6
Jose	*Noche*	7
Pedro	*Noche*	8

1. Proponer un algoritmo de ordenamiento `proc ordenaPlanilla`(`inout` p : `Array`⟨*alumno*⟩) para resolver el problema descripto anteriormente y cuya complejidad temporal sea $O(n)$ en el peor caso, donde n es la cantidad de elementos del arreglo. Justificar.
2. Modificar la solución del inciso anterior para funcionar en el caso que `Turno` sea un tipo enumerado con más elementos (donde la cantidad de los mismos sigue estando acotada y el órden final está dado por el órden de los valores en el enum. Puedo hacer **for** t **in** `Turno`).
3. ¿La cota $O(n)$ contradice el "lower bound" sobre la complejidad temporal en el peor caso de los algoritmos de ordenamiento? (El Teorema de "lower bound" establece que todo algoritmo general de ordenamiento tiene complejidad temporal $\Omega(n \log n)$.) Explique su respuesta.

Ejercicio 11. Sea $A[1 \dots n]$ un arreglo de números naturales *en rango* (cada elemento está en el rango de 1 a k, siendo k alguna constante). Diseñe un algoritmo que ordene esta clase de arreglos en tiempo $O(n)$. Demuestre que la cota temporal es correcta.

Ejercicio 12. Se desea ordenar los datos generados por un sensor industrial que monitorea la presencia de una sustancia en un proceso químico. Cada una de estas mediciones es un número entero positivo. Dada la naturaleza del proceso se sabe que, dada una secuencia de n mediciones, a lo sumo $\lfloor \sqrt{n} \rfloor$ valores están fuera del rango $[20, 40]$.

Proponer un algoritmo $O(n)$ que permita ordenar ascendentemente una secuencia de mediciones y justificar la complejidad del algoritmo propuesto.

Ejercicio 13. Se tiene un arreglo $A[1 \dots n]$ de T, donde T son tuplas $\langle c_1 : \mathbb{N} \times c_2 : string[\ell] \rangle$ y los $string[\ell]$ son *strings* de longitud máxima ℓ. Si bien la comparación de dos $\mathbb{N}$ toma $O(1)$, la comparación de dos $string[\ell]$ toma $O(\ell)$. Se desea ordenar A en base a la segunda componente y luego la primera.

1. Escriba un algoritmo que tenga complejidad temporal $O(n\ell + n\log(n))$ en el peor caso. Justifique la complejidad de su algoritmo.
2. Suponiendo que los naturales de la primer componente están acotados, adapte su algoritmo para que tenga complejidad temporal $O(n\ell)$ en el peor caso. Justifique la complejidad de su algoritmo.

Ejercicio 14. Se tiene un arreglo A de n números naturales y un entero k. Se desea obtener un arreglo B ordenado de $n \times k$ naturales que contenga los elementos de A multiplicados por cada entero entre 1 y k, es decir, para cada $1 \leq i \leq n$ y $1 \leq j \leq k$ se debe incluir en la salida el elemento $j \times A[i]$. Notar que podría haber repeticiones en la entrada y en la salida.

a) Implementar la función

$$\texttt{proc ordenarMúltiplos}(\text{in } A : \text{Array}\langle\mathbb{N}\rangle, \in k : \mathbb{N}) \ : \text{Array}\langle\mathbb{N}\rangle$$

que resuelve el problema planteado. La función debe ser de tiempo $O(nk \log n)$, dónde n es el tamaño del arreglo.

b) Calcular y justificar la complejidad del algoritmo propuesto.

Ejercicio 15. Se tiene un arreglo A de n números naturales. Sea $m := \text{máx}\{A[i] : 1 \leq i \leq n\}$ el máximo del arreglo. Se desea dar un algoritmo que ordene el arreglo en $O(n \log m)$, utilizando únicamente arreglos y variables ordinarias (i.e., sin utilizar listas enlazadas, árboles u otras estructuras con punteros).

a) Implementar la función

$$\texttt{proc raroSort} \ (\text{in A: Array}\langle\mathbb{N}\rangle) : \text{Bool} \ \{$$

que resuelve el problema planteado. La función debe ser de tiempo $O(n \log m)$, dónde n es el tamaño del arreglo y $m = \text{máx}\{A[i] : 1 \leq i \leq n\}$.

b) Calcular y justificar la complejidad del algoritmo propuesto.

Ejercicio 16. *Arreglos de enteros.*

1. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n en tiempo $O(n)$.
2. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n^2 en tiempo $O(n)$. *Pista: Usar varias llamadas al ítem anterior.*
3. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n^k para una constante arbitraria pero fija k.
4. ¿Qué complejidad se obtiene si se generaliza la idea para arreglos de enteros no acotados?

Ejercicio 17. Se nos pide ayudar a un herborista que quiere poder organizar sus ingredientes para determinar qué hierbas le conviene recolectar. Para ello cuenta con su propio inventario. Como no es una persona muy organizada, puede tener distintas hierbas del mismo tipo en distintas alacenas o cofres. Luego de realizar una inspección de su lugar de trabajo, nos entrega una secuencia de n tuplas que constan de una hierba, identificada por su nombre, y la cantidad que se encontró. El nombre de cada hierba tiene como máximo 100 caracteres, de acuerdo al estándar de la Organización Mundial de Herboristas. El herborista cuenta a su vez con su libro de creaciones, que le permite saber en cuántas recetas se utiliza cada hierba.

Se necesita saber cuáles son las hierbas que se usan en más creaciones y, en caso de empate, deberían aparecer primero aquellos de las que tiene menos reservas. La complejidad esperada en el *peor caso* es de $O(n + h\ log(h))$, donde h es la cantidad de hierbas distintas con las que cuenta el herborista.

```
proc Recolectar(in s:Vector<tupla<string,int>>, in u:Diccionario<string,int>):Vector<string>
```

Ejemplo:

```
stock = [ ("Diente de León", 10), ("Menta", 4), ("Margarita", 13),
    ("Lavanda", 12), ("Diente de León", 5), ("Margarita", 6) ]

usos = {"Diente de León": 5, "Menta": 1, "Margarita": 3, "Lavanda": 5}

Recolectar(stock, usos) = ["Lavanda", "Diente de León", "Margarita", "Menta"]
```

Los primeros son la lavanda y el diente de león porque ambos tiene 5 usos, pero aparece primera la lavanda porque hay menos stock. En tercer lugar tenemos la margarita que tiene 3 usos. Finalmente, en último lugar está la menta, que tiene un solo uso.

a) Se pide escribir el algoritmo de `Recolectar`. Justificar **detalladamente** la complejidad y escribir todas las suposiciones sobre las implementaciones de las estructuras usadas, entre otras.

b) ¿Cuál sería el *mejor caso* para este problema? ¿Cuál sería la cota de complejidad más ajustada?

Ejercicio 18. Dado un arreglo de pares `<estudiante, nota>`, queremos devolver un arreglo de pares `<estudiante, promedio>`, ordenado en forma descendente por promedio y, en caso de empate, por número de libreta.

Los estudiantes se representan con un número de libreta (puede considerar que es un número de a lo sumo 10 dígitos), y la nota es un número entre 1 y 100. La cantidad de estudiantes se considera no acotada.

Se pide dar un algoritmo que resuelva el problema con complejidad $O(n + m \log m)$, donde n es la cantidad total de notas (es decir, el tamaño del arreglo de entrada), y m es la cantidad total de estudiantes.

Ejemplo:

```
Entrada: <12395, 72>, <45615, 81>, <12395, 94>, <45615, 100>, <78920, 81>,
         <12395, 90>, <45615, 71>, <78920, 50>
Salida: <12395, 85.3333>, <45615, 84>, <78920, 65.5>
```

a) Describa cada etapa del algoritmo con sus palabras, y justifique por qué cumple con las complejidades pedidas.

b) Escriba el algoritmo en pseudocódigo. Puede utilizar (sin reescribir) todos los algoritmos y estructuras de datos vistos en clase.

Ejercicio 19. Se tienen dos arreglos de enteros, el primero contiene los elementos a ordenar en la salida y el segundo determina un criterio de ordenamiento.

```
proc OrdenarSegunCriterio(in s: Array<int>, in crit: Array<int>) : Array<int>
```

El usuario quiere reordenar s teniendo en cuenta el orden de aparición de los elementos definidos en *crit*. Los elementos que están presentes en s, pero no están presentes en *crit*, deben encontrarse al final del arreglo *res* y estar en orden creciente. El arreglo *crit* es de tamaño menor o igual que s, no tiene repetidos y puede contener elementos extra que no son parte de s.

A continuación se presenta un ejemplo:

```
s    = [5,9,3,7,3,4,3,5,1,8,4]
crit = [3,5,2,7,6]

OrdenarSegunCriterio(s, crit) = [3,3,3,5,5,7,1,4,4,8,9]
```

Como en *crit* encontramos primero a 3, luego 5 y detrás 7, todas las apariciones de 3 aparecen primeras, seguidas por las apariciones de 5 y finalmente las de 7. Como 2 y 6 no están en s, son ignorados. Luego, se encuentran todos los elementos de s que no sean 3, 5 o 7 en orden ascendente.

Se pide dar un algoritmo que haga lo requerido con complejidad de $O(n.log(n))$ siendo n la cantidad de elementos de s.