2+4+2
3+3+4

$\frac{8}{3} \to 2,$

$\frac{10}{3} \to 3,\ldots$

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2133 — Estructuras de Datos y Algoritmos 2'2022

Interrogación 1

9 de septiembre de 2022

Condiciones de entrega. Debe entregar solo 3 de las siguientes 4 preguntas.

Puntajes y nota. Cada pregunta tiene 6 puntos más un punto base. La nota de la interrogación será el promedio de las notas de las 3 preguntas entregadas.

Uso de algoritmos. De ser necesario, en sus diseños puede utilizar llamados a cualquiera de los algoritmos vistos en clase. No debe demostrar la correctitud o complejidad de estos llamados, salvo que se especifique lo contrario.

1. Análisis de algoritmos

Para ordenar una secuencia de datos implementada como arreglo A se propone el siguiente algoritmo `CocktailShakerSort` definido junto a sus subrutinas

```
input : Arreglo A[0,...,n-1]
CocktailShakerSort (A):
1   i ← 0
2   f ← n − 2
3   while i ≤ f :
4       k ← LeftRightShake(A, i, f)
5       f ← k − 1
6       t ← RightLeftShake(A, i, f)
7       i ← t + 1
```

```
input : Arreglo A[0,...,n-1], índices i, f
LeftRightShake (A, i, f):
1   j ← i
2   for m = i ... f :
3       if A[m] > A[m+1] :
4           A[m] ⇄ A[m+1]
5           j ← m
    return j

RightLeftShake (A, i, f):
1   j ← f
2   for m = f ... i :        ▷ loop decreciente
3       if A[m] > A[m+1] :
4           A[m] ⇄ A[m+1]
5           j ← m
    return j
```

(a) [1 pto.] Demuestre que los tres algoritmos mostrados terminan en tiempo finito.

(b) Demuestre que el algoritmo `CocktailShakerSort` ordena.

- [2 ptos.] Pruebe primero que `LeftRightShake` cumple su objetivo. *Hint:* luego de la iteración de valor m del **for**, ¿qué satisface el tramo $A[j \ldots m+1]$?
- [1 pto.] Pruebe que `CocktailShakerSort` ordena, dado que `LeftRightShake` y `RightLeftShake` cumplen su objetivo.

(c) [1 pto.] Determine el mejor caso de `CocktailShakerSort`. Explique brevemente su respuesta.

(d) [1 pto.] Determine su complejidad en el mejor caso.

2. Diseño de algoritmos

Un archivo contiene datos de todos los estudiantes que han rendido cursos del DCC desde 1980 hasta la fecha. El formato cada registro en el archivo es

RUT	primer_apellido	segundo_apellido	nombre

y los registros se encuentran ordenados por `RUT`.

(a) [3 ptos.] Proponga el pseudocódigo de un algoritmo para ordenar los registros alfabéticamente, i.e. según (`primer_apellido`, `segundo_apellido`, `nombre`). Especifique qué estructura básica usará (listas o arreglos). Si p es un registro, puede acceder a sus atributos con `p.primer_apellido`, `p.nombre`, etc. Además, puede asumir que todo algoritmo de ordenación visto en clase puede ordenar respecto a un atributo específico.

(b) [2 ptos.] Determine la complejidad de peor caso de su algoritmo en función del número de estudiantes en el archivo.

(c) [1 pto.] Le informan que, si bien en el archivo hay primeros apellidos repetidos, la cantidad de repeticiones por apellido es muy baja (#repeticiones < 20). ¿Puede proponer alguna mejora a su algoritmo utilizando esta información? Justifique.

3. Estrategias algorítmicas

Dada una secuencia $A[0 \ldots n-1]$ de números enteros, se define un **índice mágico** como un índice $0 \leq i \leq n-1$ tal que $A[i] = i$. Por ejemplo, en la siguiente secuencia

-7	3	2	5	-1	15	7	12	6	9	3
0	1	2	3	4	5	6	7	8	9	10

existen dos índices mágicos: el 2 y el 9.

Dada una secuencia $A[0 \ldots n-1]$ **ordenada**, sin elementos repetidos e implementada como arreglo,

(a) [4 ptos.] Proponga el pseudocódigo de un algoritmo que retorne un índice mágico en A si existe y que retorne `null` en caso contrario. Su algoritmo debe ser más eficiente que simplemente revisar el arreglo elemento por elemento, i.e. mejor que $\mathcal{O}(n)$. *Hint:* utilice la estrategia dividir para conquistar.

(b) [2 ptos.] Determine la complejidad de peor caso de su algoritmo.

4. Modificación de algoritmos

(a) [2 ptos.] Considere el algoritmo `MergeThree` que toma como argumento 3 secuencias ordenadas, en lugar de 2 como el algoritmo `Merge` visto en clases. Su output sigue siendo una nueva secuencia ordenada con los elementos de las subsecuencias. Si diseñamos una versión de `MergeSort` que utilice `MergeThree`, ¿cuál es la complejidad asintótica de esta nueva versión en función de la cantidad n de elementos en la secuencia inicial? *Hint:* Puede utilizar los supuestos que necesite sobre n para simplificar el análisis.

(b) [4 ptos.] Recuerde que dado un arreglo A, una inversión es un par de índices (i, j) tales que $i < j$ y $A[i] > A[j]$. Proponga el pseudocódigo de un algoritmo que determine el número de inversiones en un arreglo de n elementos en tiempo $\mathcal{O}(n \log(n))$ en el peor caso. *Hint:* modifique la versión de `Merge`/`MergeSort` vista en clases. No necesita demostrar su complejidad.