Algoritmos y Estructuras de Datos
Segundo Recuperatorio – Martes 10 de Diciembre de 2024

#Orden	Libreta	Apellido y Nombre	E1	E2	E3	E4	Nota Final

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, más los dos apuntes de la materia
- El parcial se aprueba con 50 puntos y hay que tener 60 puntos de promedio entre los dos parciales

E1. Complejidad (20 pts)

a) Indique si las siguientes afirmaciones son verdaderas o falsas. En caso de ser **verdadera**, justifique formalmente. En caso de ser **falsa**, dé un contraejemplo claro que respalde su análisis. Sean $f, g: \mathbb{Z} \to \mathbb{Z}$

I) Si $f \times g \in \Omega(g)$, entonces f es una función constante distinta de 0
II) Si $O(g) \cap O(f) \neq \emptyset$, entonces $f \in O(g) \wedge g \in O(f)$

b) Sea A un algoritmo que recorre un arreglo (comenzando por la primera posición) hasta encontrar un elemento e. Sea n el tamaño del arreglo, elija la o las opciones que sean correctas. **No es necesario justificar.**

I) El peor caso es $O(n)$
II) El mejor caso es $\Omega(n)$
III) El mejor caso es cuando el arreglo está vacío
IV) El peor caso es $O(n^2)$
V) El mejor caso es $\Theta(n)$
VI) El peor caso es cuando e está al final del arreglo

E2. Rep&Abs (20 pts)

Ingrediente, *Receta* es $\mathbb{Z}$

```
TAD Cocina {
obs recetario: dict⟨Receta, conj⟨Ingrediente⟩⟩
obs stock: dict⟨Ingrediente, int⟩
proc iniciarActividad() : Cocina
    asegura(Ambos diccionarios comienzan vacíos)
proc nuevaReceta(inout c : Cocina, in r : Receta, in is : conj⟨Ingrediente⟩)
    requiere(La receta no está registrada previamente y el conjunto no es vacío)
    asegura(Se registra la receta con los ingredientes requeridos)
proc comprarIngrediente(inout c : Cocina, in i : Ingrediente)
    asegura(Se le agrega una existencia al ingrediente comprado)
}
```

```
Módulo CocinaImpl implementa Cocina <
  var ingredientesPorReceta: Diccionario<Receta, Conjunto<Ingrediente>>
   // Todas las recetas con los ingredientes que son necesarios. Cada ingrediente consume una sola existencia
  var existencias: Array<Ingrediente>
   // Todos los ingredientes que se poseen. Cada ingrediente aparece tantas veces como existencias se tengan
  var ingMasAbundante: ColaDePrioridadMax<Tupla<Ingrediente, int>>
   // Cola de prioridad ordenada de acuerdo a la cantidad que se tiene de cada ingrediente
  var abundancias: Diccionario<int,Conjunto<Ingrediente>>
   // Todas las cantidades con el conjunto de los ingredientes que tienen esas existencias
>
```

a) Indique si cada una de las sentencias propuestas pertenecen al invariante de representación para la estructura propuesta

I) Dadas dos recetas distintas en **ingredientesPorReceta** sus conjuntos de ingredientes son disjuntos
II) Todo ingrediente asociado a una receta en **ingredientesPorReceta** debe estar presente en **existencias**
III) Todas las claves registradas en **abundancias** son mayores que cero
IV) Dados dos cantidades registradas en **abundancias** sus conjuntos de ingredientes son disjuntos
V) Todo ingrediente presente en **existencias** debe ser ingrediente de alguna receta registrada en **ingredientesPorReceta**

b) Proponga todas las sentencias en castellano que falten para incluir todas las restricciones correspondientes a **ingMasAbundante** en el módulo

c) Escribir la función de abstracción en lógica de primer orden

E3. Diseño (30 pts)

Una discográfica quiere organizar los conciertos de sus artistas para el próximo año (los 365 días de 2025). Cuando un artista decide hacer un concierto, elige primero un día y luego se le asigna el estadio con mayor capacidad disponible ese día. Cada estadio sólo puede tener un show por día y cada artista sólo puede hacer un show por día, por lo que al momento de elegir día el artista debe estar libre y debe haber al menos un estadio disponible. Cuando un artista quiere cancelar un concierto, debe proponer a otro artista para que lo reemplace. En caso de que el reemplazo no esté disponible, el concierto se cancela y el estadio vuelve a estar disponible. Sean n la cantidad de artistas y m la cantidad de estadios y los tipos *Estadio*, *Capacidad*, *Artista* y *Dia* un reemplazo para *int*.

- `proc ProgramarConcierto`(inout dg : $Discografica$, in d : Dia, in a : $Artista$)
  - **Descripción**: Programa un concierto para ese día en el estadio con más capacidad disponible
  - **Requiere**: El artista y algún estadio están libres ese día
  - **Complejidad**: $O(log(m) + log(n))$
- `proc CancelarConcierto`(inout dg : $Discografica$, in d : Dia, in a : $Artista$, $\in b$: $Artista$)
  - **Descripción**: Artista a cancela el concierto del día d, con b como reemplazo. Si b está ocupado ese día, el concierto se cancela y el estadio se libera
  - **Requiere**: El artista a tiene un concierto programado el día d
  - **Complejidad**: $O(log(m) + log(n))$
- `proc ListadoConciertos`(inout dg : $Discografica$, in a : $Artista$) : $ListaEnlazada(< Dia, Estadio, Capacidad >)$
  - **Descripción**: Devuelve el listado ordenado temporalmente de conciertos del artista a
  - **Complejidad**: $O(log(n))$
- `proc CantidadPorEstadio`(in dg : $Discografica$, in e : $Estadio$) : int
  - **Descripción**: Devuelve la cantidad de conciertos programados para el estadio e
  - **Requiere**: El estadio e está entre los estadios definidos
  - **Complejidad**: $O(log(m))$

Se pide:

a) Plantear la estructura de representación del módulo `DiscograficaImpl`. **Justificar** cómo se cumplen las complejidades requeridas de las operaciones mencionadas

b) Escribir el algoritmo de `CancelarConcierto`. Justifique **detalladamente** que se cumple la complejidad requerida

E4. Sorting (30 pts)

a) **Teórica**. Dados los algoritmos **SelectionSort** y **InsertionSort**. Supongamos que se detienen estos algoritmos en la iteración i-ésima. ¿Qué podemos decir del orden de los elementos hasta ese momento? ¿Estos algoritmos son estables?

b) Un prestamista nos contrata para ayudarle a organizar de forma eficiente su veraz. El veraz consiste de un registro donde a cada persona se le asocia un ***score*** de deuda y el **monto** adeudado hasta el momento. El *score* es un valor entero entre 0 y 100, mientras que el monto adeudado sólo se sabe que es un entero positivo expresado en USD. Se sabe además que dentro de la base del veraz, solamente el $\sqrt{n}$ deudores presentan una deuda mayor a 100.000 USD, siendo n la cantidad total de deudores registrados. Dado un arreglo que consiste en tuplas de la forma $< ID, Score, Monto >$ que representa el veraz, donde el ID (int) es un valor único que permite identificar a las personas, se pide ordenar a las personas del veraz de acuerdo al monto adeudado al sistema financiero de forma ascendente y, en caso de empate, según el *score* asignado por el sistema de forma descendente. Dar un algoritmo que resuelva el problema propuesto en $O(n)$. **Justifique** correctamente las decisiones tomadas y las complejidades obtenidas de los algoritmos utilizados.

Ejemplo

Si recibimos:

```
[ <143, 20, 500>, <37, 50, 300.000>, <5, 33, 500>, <7, 90, 300.000>, <999, 13, 1.542>]
```

Deberíamos devolver:

```
[ <7, 90, 300.000>, <37, 50, 300.000>, <999, 13, 1.542>, <5, 33, 500>, <143, 20, 500>]
```