Algoritmos y Estructuras de Datos II – 6 de Mayo de 2026

Examen Parcial Teórico-Práctico

Alumno: ..... Email:

Siempre se debe explicar la solución. Una respuesta correcta no es suficiente si no viene acompañada de una justificación lo más clara y completa posible. Los algoritmos no deben escribirse utilizando código c o de bajo nivel, sino el código de la materia y evitando la utilización innecesaria de punteros. Realizar cada ejercicio en HOJAS SEPARADAS y NOMBRADAS. La no observación de estas recomendaciones resta puntaje.

1. (TADs) Considere la siguiente especificación del tipo Conjunto de elementos de algún tipo T.

```
spec Set of T where

constructors
    fun empty_set() ret s : Set of T
    {- crea un conjunto vacío. -}

    proc add (in e : T, in/out s : Set of T)
    {- agrega el elemento e  al conjunto s. -}

destroy
    proc destroy (in/out s : Set of T)
    {- Libera memoria en caso que sea necesario. -}

operations
    fun is_empty_set(s : Set of T) ret b : bool
    {- Devuelve True si s  es vacío. -}

    fun cardinal(s : Set of T) ret n : nat
    {- Devuelve la cantidad de elementos que tiene s -}

fun member(e : T, s : Set of T) ret b : bool
{- Devuelve True si el elemento  e  pertenece al conjunto s -}

proc inters (in/out s : Set of T,in s0 : Set of T)
{- Elimina de  s  todos los elementos que NO pertenecen a  s0 -}

{- PRE:  not is_empty_set(s) -}
fun get(s : Set of T) ret e : T
{- Devuelve algún elemento (cualquiera) de s -}

proc elim (in/out s : Set of T,in e : T)
{- Elimina el elemento e del conjunto s en caso que esté. -}

proc union (in/out s : Set of T,in s0 : Set of T)
{- Agrega a s todos los elementos de s0 -}

proc diff (in/out s : Set of T,in s0 : Set of T)
{- Elimina de s todos los elementos de s0 -}

    fun copy_set(s1 : Set of T) ret s2 : Set of T
    {- Crea un nuevo conjunto s2 con todos los elementos de s1 -}
```

(a) Implementá los constructores del TAD Conjunto de elementos de tipo T, y las operaciones member, elim e inters, utilizando la siguiente representación:

```
implement Set of T where

type Set of T = tuple
                    elems : array[0..N-1] of T
                    size : nat
                end tuple
```

¿Qué orden de complejidad tienen estas operaciones?

¿Existe alguna limitación con esta representación de conjuntos? En caso afirmativo indicá si algunas de las operaciones o constructores tendrán alguna precondición adicional.

NOTA: Si necesitás alguna operación extra para implementar lo que se pide, debes implementarla también.

(b) Utilizando el tipo **abstracto** Conjunto de elementos de tipo T, implementá una función que reciba un conjunto de enteros s, un número entero i, y obtenga el entero perteneciente a s que está *más cerca* de i, es decir, un $j \in s$ tal que para todo $k \in s$, $|j - i| <= |k - i|$. Por ejemplo si el conjunto es $1, 5, 9$, y el entero 7, el resultado puede ser 5 o 9.

2. (a) Dado el siguiente arreglo, mostrá cómo queda el mismo luego de cada modificación que realiza el algoritmo **merge sort**.

$$a = [10, 2, 4, 3, 12, 5, 3]$$

Además da las secuencias de llamadas al procedimiento *merge_sort_rec* indicando correctamente sus argumentos.

(b) Escribí una variante del algoritmo de ordenación por inserción que vaya ordenando desde la última celda del arreglo hacia la primera. El resultado debe ser el mismo, es decir, el arreglo resultante debe estar ordenado en forma creciente, pero el modo de hacerlo debe ser diferente: en cada paso se debe *insertar* el elemento correspondiente, moviéndolo hacia la derecha hasta que sea menor o igual que el siguiente. ¿Qué orden tiene el algoritmo implementado?

3. Para cada uno de los siguientes algoritmos determinar **por separado** cada uno de los siguientes incisos.

(a) ¿Qué hace? ¿Cuáles son las precondiciones necesarias para haga eso?

(b) Si le aplicamos el procedimiento a la lista que tiene los elementos $[1, 2, 3, 4, 5]$ ¿cómo es la lista resultante?

(c) ¿Cómo lo hace?

(d) El orden del algoritmo, analizando los distintos casos posibles.

```
proc p (in/out l: list of T)
    var a, b: pointer to (node of T)
    a := l
    while a ≠ null
        and a→next ≠ null do
        b := a→next
        a→next := b→next
        free(b)
        a := a→next
    od
end proc
```

donde los tipos node y list se definen como sigue

```
type node of T = tuple
                    value: T
                    next: pointer to (node of T)
                end
type list of T = pointer to (node of T)
```