Algoritmos y Estructuras de Datos II - 2 de Mayo de 2022

Primer Parcial

Nombre:.....

¡ENTREGAR CADA EJERCICIO EN HOJAS SEPARADAS!

Siempre explicar la solución. Una respuesta correcta no es suficiente si no viene acompañada de una justificación que demuestre que la misma ha sido comprendida. En el ejercicio de implementación debe utilizarse el lenguaje del teórico de la materia. **La utilización de código C influirá negativamente.**

1. Se conoce con el nombre de 'cocktail sort' a una variación del algoritmo de ordenación 'selection sort' que consiste en obtener en cada pasada del arreglo no sólo el elemento mínimo del segmento correspondiente, sino el mínimo y el máximo, intercambiándolos luego por las posiciones primera y última del segmento, respectivamente. Por ejemplo si tenemos el arreglo '[8,5,18,3,4]' (con posiciones de 1 a 5), el algoritmo en su primera pasada encontraría que el elemento mínimo está en la posición 4, y el máximo en la posición 3, que al intercambiarlos el arreglo resultante sería '[3,5,4,8,18]'.

   Se pide:

   (a) Explicá con palabras claras cómo implementarás el algoritmo.

   (b) Implementá de manera precisa el 'cocktail sort' en el lenguaje de la materia.

   (c) Mostrá cómo funciona el algoritmo corriendo "a mano" el algoritmo sobre el arreglo '[5,12,4,8,2,7]'.

2. Dados el siguiente procedimiento

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

donde los tipos node y list se definen como sigue

```
type node = tuple
                value: elem
                next: pointer to node
            end
type list = pointer to node
```

   (a) Explicá qué hace el procedimiento p.

   (b) ¿Cuál es el orden del procedimiento p? Justificá la respuesta.

   (c) Si se llama al procedimiento p con una lista que tiene los valores 1, 2, 3, 4, 5, 6 y 7, en ese orden, ¿qué valores tendrá la lista luego de la llamada?

3. Dada la siguiente función:

```
fun f(n: nat) ret m: nat
    if n ≤ 1 then m:= 2 * n
    else m:= 1
         for i:= n downto 1 do m:= n * m od
         m:= 3 * f(n div 2)
    fi
end fun
```

   (a) ¿Cuántas llamadas recursivas a f(n div 2) se realizan durante la ejecución de f(n)?

   (b) Expresá la ecuación de recurrencia en función de la cantidad de asignaciones a la variable m.

   (c) Calculá el orden de asignaciones a la variable m.

4. Considere la siguiente especificación del tipo Listas de algún tipo T.

```
spec List of T where

constructors
      fun empty() ret l : List of T
      {- crea una lista vacía. -}
```

proc addl (in e : T, in/out l : List of T)
{- *agrega el elemento* e *al comienzo de la lista* l. -}

destroy

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

operations

fun is_empty(l : List of T) ret b : bool
{- *Devuelve* True *si* l *es vacía.* -}

fun head(l : List of T) ret e : T
{- *Devuelve el primer elemento de* l -}
{- PRE: not is_empty(l) -}

proc tail(in/out l : List of T)
{- *Elimina el primer elemento de* l -}
{- PRE: not is_empty(l) -}

proc addr (in/out l : List of T,in e : T)
{- *agrega el elemento* e *al final de* l -}

fun length(l : List of T) ret n : nat
{- *Devuelve el largo de* l -}

proc concat(in/out l : List of T,in l0 : List of T)
{- *Agrega al final de* l *todos los elementos de* l0 *en el mismo orden.*-}

fun index(l : List of T,n : nat) ret e : T
{- *Devuelve el n-ésimo elemento de la lista* l -}
{- PRE: length(l) ¿ n -}

proc take(in/out l : List of T,in n : nat)
{- *Deja en* l *sólo los primeros* n *elementos, eliminando el resto* -}

proc drop(in/out l : List of T,in n : nat)
{- *Elimina los primeros* n *elementos de* l -}

fun copy_list(l1 : List of T) ret l2 : List of T
{- *Copia todos los elementos de* l1 *en la nueva lista* l2 -}

- Implementá una función que reciba dos listas de enteros y devuelva otra lista que contenga en las posiciones pares todos los elementos de la primera lista (en el mismo orden), y en las posiciones impares todos los elementos de la segunda lista (en el mismo orden).
  Para ello utilizá el tipo **abstracto**, sin acceder a su representación interna.