Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2612 - Algoritmos y Estructuras II
Enero-Marzo 2011

Carnet:_____

Nombre:_____

Examen I

(25 puntos)

Pregunta 0	Pregunta 1	Pregunta 2	Pregunta 3	Total
4 puntos	5 puntos	6 puntos	10 puntos	25 puntos

Pregunta 0 - 4 puntos

Esta pregunta consta de diez (4) subpreguntas de selección, numeradas de 0.0 a 0.3. Cada una de las subpreguntas viene acompañada de cuatro posibles respuestas (a, b, c, d), entre las cuales sólo **una** es correcta. Ud. deberá marcar completamente y sin posibilidad de confusión la opción que considere correcta o, si desea no contestar la subpregunta, no marcar ninguna de las opciones, si Ud. marca más de una opción su respuesta será considerada omitida.
Cada una de las subpreguntas tiene un valor de un punto (1). Tres subpreguntas incorrectas eliminan una correcta. Las subpreguntas omitidas (en las que se han dejado todas las opciones sin marcar o se han marcado más de una opción) no suman ni restan puntos.

0.0. En esta y las subpreguntas siguientes, se utilizará Ω y O, respectivamente, para la complejidad de mejor y peor caso de un algoritmo. Además, se querrá siempre para mejor caso la mayor cota inferior conocida, y para peor caso la menor cota superior conocida.

Considere el algoritmo de ordenamiento por mezcla (*Merge Sort*), puede considerar la versión presentada en el capítulo 15 del Hume, la versión propuesta en el capítulo 2 del Cormen, o la versión vista en clase. Las cotas de complejidad para mejor y peor caso de este algoritmo son:

a) $\Omega(1)$ y $O(n^2)$.
b) $\Omega(n)$ y $O(n^2)$.
c) $\Omega(n^2)$ y $O(n^2)$.
d) Ninguna de los anteriores

0.1. Tomando en cuenta la aclaratoria inicial de la subpregunta 0.0, considere el algoritmo de ordenamiento por inserción (*Insertion Sort*), puede considerar la versión presentada en el capítulo 2 del Cormen, o la versión vista en clase. Las cotas de complejidad para mejor y peor caso de este algoritmo son:

a) $\Omega(1)$ y $O(n^2)$.
b) $\Omega(n)$ y $O(n^2)$.
c) $\Omega(n^2)$ y $O(n^2)$.
d) Ninguna de los anteriores

0.2. Tomando en cuenta la aclaratoria inicial de la subpregunta 0.0, considere el siguiente algoritmo:

```
const n : int;
var a : array[0..n) of int;
var i : int;

i := 0;

do (i < n - 1 /\ a[i] < a[i+1]) ->
    var j : int;

    j := i + 1;

    do (j < n /\ a[i] =/= a[j]) ->
        a[j] := a[i] + j;
        j := j + 1
    od;
    i := i + 1
od
```

donde `=/=` representa diferencia ($\neq$).

Las cotas de complejidad para mejor y peor caso de este algoritmo son:

a) $\Omega(1)$ y $O(n^2)$.

b) $\Omega(n)$ y $O(n^2)$.

c) $\Omega(n^2)$ y $O(n^2)$.

d) Ninguna de los anteriores

0.3. Tomando en cuenta la aclaratoria inicial de la subpregunta 0.0, considere el algoritmo de ordenamiento por selección (*Selection Sort*), puede considerar la versión presentada en el capítulo 15 del Hume, la versión propuesta en el capítulo 2 del Cormen, o la versión planteada en clase. Las cotas de complejidad para mejor y peor caso de este algoritmo son:

a) $\Omega(1)$ y $O(n^2)$.

b) $\Omega(n)$ y $O(n^2)$.

c) $\Omega(n^2)$ y $O(n^2)$.

d) Ninguna de los anteriores

Pregunta 1 - 5 puntos

Se desea que Ud. demuestre que Ω es transitiva. Esto es, demostrar que para todas funciones asintóticamente no-negativas f(n), g(n) y h(n), se cumple que: si $f(n) = \Omega(g(n))$ y $g(n) = \Omega(h(n))$ entonces $f(n) = \Omega(h(n))$.

Nota: En esta demostración sólo es valido utilizar la definición básica de Ω presentada en el capítulo 3 del Cormen y dada en clase. Esto es, no puede utilizar otras propiedades conocidas de notación asintótica.

Pregunta 2 - 6 puntos

Se desea realizar una implementación del algoritmo de ordenamiento rápido (*QuickSort*) con un peor caso con mejor desempeño que la versión diseñada por Hoare en 1960, el peor caso de esta versión es $\Theta(n^2)$. Para esto se propone dividir el segmento a ordenar en 3 partes en lugar de 2 (como hace la versión usual de *QuickSort*), para ello el procedimiento iterativo auxiliar *partition* deberá utilizar dos pivotes, y devolver la posición de estos dos pivotes en el arreglo, además de haber reorganizado los elementos del arreglo de tal forma que, siendo el primer pivote menor o igual al segundo, a la izquierda del primer pivote estén los elementos menores a éste, entre el primer y segundo pivote se ubiquen los elementos mayores o iguales al primer pivote pero menores o iguales al segundo, y a la derecha del segundo pivote estarán los elementos mayores al segundo pivote. *partition* trabajará sobre segmentos de al menos dos elementos, y los pivotes, por supuesto, son elementos que están en el segmento, por ejemplo si el primer elemento del segmento es `p` y el último es `u`, el primer pivote será `p min u`, y el segundo pivote será `p max u`.

Sabiendo que se cuenta con una implementación del procedimiento auxiliar *partition* a utilizar con una complejidad asociada de $\Theta(n)$, se desea que Ud. escriba la recurrencia correspondiente al peor caso de la versión de *QuickSort* arriba descrita, y determine la cota asintótica exacta de ésta justificando su respuesta mediante alguno de los métodos de resolución de recurrencias vistos en clase y presentes en el capítulo 4 del Cormen. Adicionalmente indique si se logró el objetivo de mejorar el desempeño del algoritmo para el peor caso.

Pregunta 3 - 10 puntos

Se desea que Ud. construya un procedimiento que implemente una versión de Ordenamiento por Selección (*Selection Sort*), según las pautas dadas a continuación:

La especificación del procedimiento es:

proc `seleccion (` **in** `n : int;` **in-out** `a : array[0..n) of T)`
{ **Pre:** $n \geq 0$ }
{ **Post:** $ordenado(a[0..n)) \land bag(a[0..n)) = bag(a_0[0..n))$ }

Donde la función auxiliar `bag` se define como:
$bag(a[b..c)) = \biguplus\ i : b \leq i < c : a[i]\ \biguplus$

Y ésta es usada para denotar el multiconjunto asociado a un arreglo, y se hace uso de ella en la postcondición para indicar que los elementos del arreglo luego de la ejecución del procedimiento deben ser exactamente los mismos que los contenidos en el arreglo antes de la ejecución del procedimiento.

También se utiliza el predicado `ordenado` que indica que el arreglo dado como argumento se encuentra ordenado de forma no decreciente:
$ordenado(a[b..c)) = (\forall\ i : b \leq i < c - 1 : a[i] \sqsubseteq a[i+1])$

Donde $\sqsubseteq$ denota la relación "menor o igual" entre elementos de tipo `T`.

Se desea que Ud. construya una implementación iterativa de este procedimiento, cuya iteración principal debe ser construida según el siguiente invariante, especificado parcialmente:
$0 < k \leq n \land ordenado(a[k..n)) \land bag(a[0..n)) = bag(a_0[0..n)) \land \ldots$

Siendo `k` una nueva variable entera.

De acuerdo con todo lo señalado, haga lo siguiente:

- Señale si el recorrido que la variable `k` debe hacer sobre el arreglo en la iteración principal debe ser de izquierda a derecha o de derecha a izquierda según el invariante dado, y explique **muy brevemente** por qué.
- Complete la implementación utilizando iteración (no puede utilizar recursión), señalando tanto el invariante completo como la cota variante de terminación correspondientes a todas las iteraciones que utilice.