Universidad de Costa Rica
CI-1221 Estructuras de Datos y Análisis de Algoritmos
I Ciclo 2010

I EXAMEN

Martes 6 de abril, 5 a 7:30 p.m.

El examen consta de 166 puntos pero no se reconocerán más de 120 (20% extra). Cada pregunta empieza con un indicador de su puntaje y del tema del que trata. Si la pregunta tiene subítemes el puntaje de cada uno de ellos es indicado al final de la pregunta. Se recomienda al estudiante echarle un vistazo a los temas de las preguntas y a su puntaje antes de empezar a resolver el examen, para así distribuir su tiempo de la mejor manera, de acuerdo a sus destrezas.

1. [16 pts.] *Matemática básica*. Muestre que

   (a) $1+2+...+n=\frac{n(n+1)}{2}$ [8 pts.]

   (b) $a^{\log_b c}=c^{\log_b a}$ [8 pts.]

2. [16 pts.] *Correctitud de un algoritmo*. La secuencia de Fibonacci se define como:

   $F_0=0,$
   $F_1=1,$
   $F_n=F_{n-2}+F_{n-1} \quad (\forall\, n>1).$

   El siguiente algoritmo computa F_n:

```
Fibonacci(n)
    if n < 2
        return n
    a = 0
    b = 1
    for i = 2 to n
        c = b + a
        a = b
        b = c
    return c
```

   Pruebe que este algoritmo es correcto mostrando claramente los pasos de inicialización [4 pts.], mantenimiento [8 pts.] y culminación [4 pts.] del ciclo `for`.

3. [39 pts.] *Solución de recurrencias*. Resuelva las siguientes recurrencias asumiendo en cada caso que $T(n)=\Theta(1)$ para $n\leq 1$.
   (a) $T(n)=8T(n/3)+c_0+c_1 n+c_2 n^2$ [4 pts.]
   (b) $T(n)=2T(2n/3)+c_0+c_1 n+c_2 n^2$ [5 pts.]
   (c) $T(n)=2T(n/2)+c_0+c_1 n$ [6 pts.]

(d) $T(n)=T(n-1)+c_0$ [8 pts.]
(e) $T(n)=T(n/2)+\lg n$ [16 pts.]

4. [12 pts.] *Montículos*. Las tres operaciones fundamentales para recorrer los nodos de un montículo son:

   $h_i(k) = 2k$ (hijo izquierdo de k)
   $h_d(k) = 2k+1$ (hijo derecho de k)
   $p(k) = \lfloor k/2 \rfloor$ (padre de k)

   Muestre cómo implementar eficientemente estas fórmulas en binario utilizando las operaciones *left-shift* (desplazamiento a la izquierda), *right-shift* (desplazamiento a la derecha) y *or* (inclusivo). [$h_i(k)$ 3 pts., $h_d(k)$ 6 pts. y $p(k)$ 3 pts.]

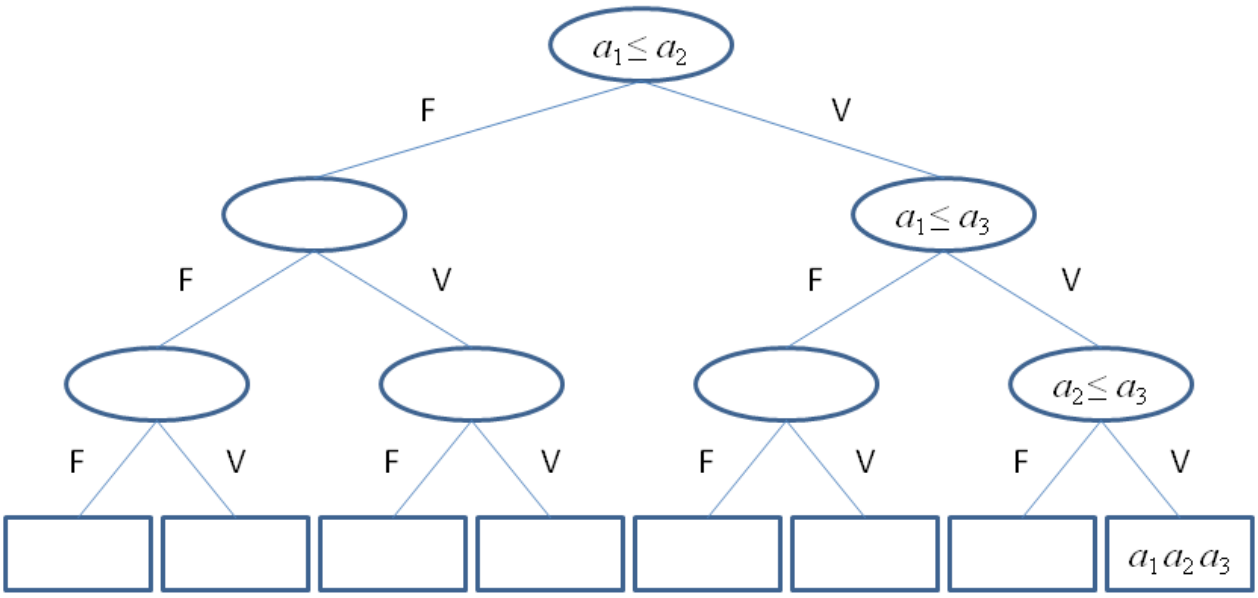
5. [37 pts.] *Simulación de ejecución de algoritmos*. Simule la ejecución de los siguientes algoritmos sobre el arreglo $A=[0,2,1,3]$ mostrando el (los) estado(s) del (los) (sub)arreglos después de cada "paso" (iteración o llamado a [sub]función del algoritmo) [4 pts. c/ simulación, excepto para ordenamiento por montículos: 4 pts. por monticularizar y 4 pts. por ordenar]. Indique además en cada "paso" el número de comparaciones realizadas entre elementos del arreglo y el número de escrituras *al arreglo* (p. ej. la escritura a una variable temporal como parte de un intercambio de posiciones no cuenta como escritura al arreglo, solo las dos escrituras hechas propiamente al arreglo), y el número total de ambas al finalizar el algoritmo [1 pto. c/ total]. Clasifique (ordene) los algoritmos de ordenamiento de acuerdo al total (la suma) de comparaciones y escrituras realizadas [1 pto.]. Indique si el mejor y peor algoritmos de acuerdo a esta clasificación coinciden con su clasificación asintótica (cuando $n\to\infty$) y si no es así, indique la(s) razón(es) por la(s) cual(es) el mejor y peor algoritmo ocuparon esa posición en este ejemplo [4 pts.].
   (a) Ordenamiento por selección
   (b) Ordenamiento por inserción
   (c) Ordenamiento por mezcla (*merge sort*)
   (d) Ordenamiento rápido (*quicksort*) utilizando el primer elemento del subarreglo como pivote.
   (e) Ordenamiento usando montículos (*heapsort*) (Para este caso puede utilizar la representación en árbol para mostrar los estados)
   (f) *Radixsort* con dígitos en binario (Como *radixsort* no hace comparaciones directas entre los elementos, para el conteo de comparaciones considere cada incremento de contador de dígitos hecho *counting sort* como una comparación [tiene sentido, ¿verdad?])
   (g) Búsqueda de la mediana con tiempo promedio $\Theta(n)$ utilizando el primer elemento del subarreglo como pivote.
   NOTAS:
      i. Si el arreglo no cambia a lo largo de varios pasos, no es necesario replicar su estado ni los conteos de operaciones siempre y cuando se explique la situación. (El uso de puntos suspensivos puede aveces ser esclarecedor aquí).
      ii. Si por la forma en que opera el algoritmo es evidente que solo parte del arreglo cambia después de un paso, se puede omitir escribir el resto del arreglo. (El uso de puntos suspensivos puede aveces ser esclarecedor aquí).
      iii. La calificación recibida por el conteo de operaciones será proporcional a $\max\{1-E, 0\}$ donde E es el error relativo:

$$E = \frac{|\#\text{ operaciones contadas} - \#\text{ operaciones real}|}{\#\text{ operaciones real}}$$

6. [15 pts.] *Quicksort*. Suponga que una compañía crea un servicio de ordenamiento de arreglos por medio de la Web. La compañía observa que su implementación de *quicksort* en promedio ordena bastante rápido arreglos por debajo de un millón de elementos. Por este motivo ofrece el servicio gratis siempre y cuando el arreglo no supere este tamaño, y cobra un cierto monto si el arreglo es más grande. Si algún malintencionado obtiene el código de *quicksort* que usa la compañía y se da cuenta que el submétodo *Partition* siempre escoge el primer elemento del (sub) arreglo como pivote podría someter una gran cantidad de arreglos ordenados (peor caso de *quicksort*) y sobrecargar enormemente el sistema.
   Describa cómo evitar que múltiples sumisiones de un arreglo en particular que no tenga elementos repetidos afecten una y otra vez el desempeño de *quicksort* [10 pts.] e implemente su idea modificando el siguiente código [5 pts.]:

```
Quicksort(A,i,j)
    if i < j
        k = Partition(A,i,j)
        Quicksort(A,i,k-1)
        Quicksort(A,k+1,j)
```

7. [11 pts.] *Árboles de decisión*. Complete el siguiente árbol de decisión correspondiente al algoritmo de ordenamiento por selección para un arreglo de tamaño 3:



8. [20 pts.] *Radixsort*. *Radixsort* ordena enteros de b bits partiéndolos en grupos de r bits y aplicando el algoritmo de ordenamiento por conteo a cada grupo, empezando por el grupo menos significativo, y continuando con el siguiente grupo más significativo en cada ocasión. La complejidad temporal del algoritmo es

$$T(n) = \Theta\left(\frac{b}{r}\,\text{máx}\{n, 2^r\}\right)$$

Muestre que si $b > \lfloor \lg n \rfloor$ lo óptimo es tomar $r = \lfloor \lg n \rfloor$.