Segundo examen Opcional
Fundamentos de Análisis y Diseño de Algoritmos
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle

Prof. Orlando Arboleda Molina

17 de Diciembre de 2005

Primera parte: Conceptos básicos [26 pts.] [1 hora]

Nombre:________________________________
Código:________________

1. **Heapssort [6 pts.]**

   Para cada una de las siguientes afirmaciones decida si es verdadera (V) o falsa (F).

   a) **[2 pts.]** El algoritmo *Heapsort* tiene como entrada un arbol, por lo tanto algunos de los subárboles (izquierdos o derechos) del mismo pueden ser nulos. ( )

   b) **[2 pts.]** En el procedimiento *Build-Heap*, se construye el montículo, en un ciclo desde length[A]/2 hasta 1 de manera decreciente. Esto es un capricho de implementacion, pues de igual forma, el montículo puede ser construido, si el ciclo fuese desde 1 hasta length[A], aplicando el procedimiento *Heapify* en cada iteración (sin importar que se procesen mas datos). ( )

   c) **[2 pts.]** Si se deseara que el algoritmo Heapsort ordenara descendentemente, es suficiente con la siguiente modificación. ( )

   ```
   HeapSortDesc(A)
       Build-Heap(A)
       for i = 2 to n − 1
           Heapify(A,i)
   ```

2. **Algoritmos de Tiempo Lineal y Ordenes estadisticos [8 pts.]**

   Para cada una de las siguientes afirmaciones decida si es verdadera (V) o falsa (F) y justifique su respuesta.

   a) **[2 pts.]** El *CountingSort* es un algoritmo por comparaciones. ( )

   b) **[2 pts.]** La complejidad de *CountingSort* siempre es $\mathcal{O}(n)$. ( )

   c) **[2 pts.]** El *BucketSort* es un algoritmo estable. ( )

   d) **[2 pts.]** El obtener el i_{esimo} orden estadistico solo puede ser obtenido $\Omega(nlgn)$, pues es necesario ordenar completamente el arreglo para obtener el i_{esimo}elemento más grande. ( )

3. **Cual es el mejor? [12 pts.]**

Se desea listar en orden ascendente los i elementos mayores de un arreglo A.

a) **[9 pts.]** Para cada uno de los métodos a continuación, calcule su complejidad en el peor caso, en función de n e i.. Para cada caso sea muy claro en la forma como presenta sus cálculos. Justifique su respuesta

1) **[3 pts.]** Ordene los números y liste los i más grandes.

2) **[3 pts.]** Construya una cola de prioridad a partir de los números y llame la función que extrae el máximo i veces.

3) **[3 pts.]** Use un algoritmo para calcular el $n - i + 1$-ésimo elemento, úselo como pivote para partición, y ordene el subarreglo que contiene los i más grandes números.

b) **[3 pts.]** Cuál de los tres es el mejor método?. Justifique con base en las complejidades indicadas previamente

Segunda parte: Algoritmos de ordenamiento [24 pts.] [1 hora 30 minutos]

Nombre:________________________________

Código:________________

4. **SelectionSort [6 pts.]**

Considere el siguiente algoritmo de ordenamiento:

```
SelectionSort(A)
for i = 1 to n − 1
    do min_ind ← i
        for j = i + 1 to n
            do if A[j] < A[min_ind]
                then min_ind ← j
        A[min_ind] ↔ A[i]
```

a) **[3 pts.]** Sea $T(n)$ el número de comparaciones entre elementos efectuadas por *SelectionSort* para entradas de tama no n en el peor caso. Cuál es el orden de $T(n)$?. Justifique

b) **[3 pts.]** Si Ud. debe escoger entre el *insertionSort* visto en clase y el *SelectionSort* descrito en este punto, cuál escogería? Justifique. **No utilice más de cinco (5) líneas**.

5. **BinaryInsertionSort [12 pts.]**

Considere el siguiente algoritmo de ordenamiento:

$BinaryInsertionSort(A)$
for $j \leftarrow 2$ **to** length$[A]$
  $\triangle A[1..j-1]$ ordenado.
  do $\triangle$ Insertar Binariamente $A[j]$ en $A[1..j-1]$
    $BinaryInsertion(A, 1, j)$

Donde $BinaryInsertion(A, i, j)$ recibe un arreglo A ordenado entre i y $j-1$, e inserta binariamente $A[j]$ en $A[i..j-1]$ de manera que al terminar $A[i..j]$ está ordenado:

$BinaryInsertion(A, i, j)$
if $i < j$
  then $ll \leftarrow A[j]$
    $m \leftarrow i + \lfloor \frac{j-i}{2} \rfloor$
    if $A[m] < A[j]$
      then $BinaryInsertion(A, m+1, j)$
      else for $k = j-1$ **downto** m
          do $A[k+1] \leftarrow A[k]$
        $A[m] \leftarrow ll$
        $BinaryInsertion(A, i, m)$
return

a) **[3 pts.]** Describa en la tabla que hay al final del presente parcial, el funcionamiento de $BinaryInsertion$, describiendo el valor de cada uno de los valores solicitados en la tabla que encontrará a continuación, para el caso en que se haga un llamado con $A = [1\ 2\ 3\ 4\ 9\ 10\ 14\ 16\ 8\ 7], i = 1, j = 9$.

No	A										i	j	ll	m
1	1	2	3	4	9	10	14	16	8	7	1	9		
2														
3														
4														
5														

b) **[6 pts.]** Sea $T(n)$ el número de comparaciones entre elementos del arreglo efectuadas, en el peor caso, por $BinaryInsertion$ para entradas de tama no n. Describa la ecuación de recurrencia para $T(n)$ y resuélvala. Cuál es, entonces, la complejidad de $BinaryInsertionSort$?

c) **[3 pts.]** Si Ud. debe escoger entre el $QuickSort$ visto en clase y el $BinaryInsertionSort$ descrito en este punto para ordenar un arreglo de n elementos, cuál escogería? Justifique su respuesta. **No utilice más de cinco (5) líneas para ello**.

6. **Ordenamiento en tiempo lineal [6 pts.]**

Describa un algoritmo para ordenar n enteros en el rango de 1 a n^2 en tiempo $\mathcal{O}(n)$.