Algorítmica
Hoja 2 de problemas – Divide y vencerás

1. Se sabe que el método de ordenación por mezcla posee una constante oculta alta, y se quiere optimizar utilizando la ordenación por selección en tamaños de vector muy pequeños. Se han programado ambos métodos de ordenación en una computadora C y se ha comprobado que la ordenación por selección posee un tiempo de ejecución $t(n)=n^2$ microsegundos, y que la mezcla de vectores se hace en tiempo de $t(n)=4n$ microsegundos. Calcular cuál es el umbral a partir del cual se debe aplicar la ordenación por selección, y escribir el pseudocódigo de la versión modificada de la ordenación por mezcla.

2. Desarrollar un procedimiento *de complejidad lineal* llamado *pivotebis* para ser usado por el algoritmo de ordenación rápida (*quicksort*) que cumpla la siguiente especificación:

```
procedimiento pivotebis (T: matriz[1..n] de entero, i, j, var k,
var l: entero)

{Este procedimiento elige como pivote el contenido de la primera
casilla, T[i], y coloca todos los elementos menores que ese pivote
antes del mismo, devuelve en k la posición de este elemento, a
continuación pone todos los elementos iguales a él, devuelve en l
la posición del último de estos elementos, y coloca detrás de este
todos los mayores que el pivote}
```

   Como ejemplo, el vector [4,5,3,4,8,5,4,8,3,2] tendría el aspecto [3,3,2,4,4,4,5,8,5,8], siendo el orden de las tres primeras casillas y el de las cuatro últimas indiferente, y siendo k=4 y l =6.

3. Escribir una versión del algoritmo de ordenación rápida (*quicksort*) que utilice el procedimiento *pivotebis* y estudiar su complejidad en el caso peor, en media y en el caso mejor. Definir con la máxima generalidad posible estos casos.

4. Desarrollar un procedimiento *de complejidad lineal* llamado *pivotebis2* para ser usado por el algoritmo de ordenación rápida (*quicksort*) que cumpla la siguiente especificación:

```
procedimiento pivotebis2 (T: matriz[1..n] de entero, i, j, var k:
entero)

{Este procedimiento elige como pivote el contenido de la casilla
situada en el punto medio y dispone todos los elementos menores al
pivote antes del mismo, el pivote a continuación y los mayores que
el pivote después}
```

   Como ejemplo, para el vector [4,5,3,4,5,8,4,8,3,2], el elemento seleccionado como pivote sería T[5]=5 y el resultado tendría el aspecto [4,5,3,4,5,3,2,5,8,8], siendo el orden de las siete primeras casillas y el de las dos últimas indiferente, y siendo k=8.

5. La versión inicial de la ordenación rápida (*quicksort*) funciona perfectamente con el procedimiento *pivotebis2*. Estudiar la complejidad de esta versión con *pivotebis2* en los casos peor, medio y mejor y definir con la máxima generalidad posible todos los casos.

6. Diseñar un algoritmo de búsqueda binaria *busbin2* que divide la tabla en dos partes de tamaños 1/3 y 2/3 respectivamente, en lugar de las habituales partes iguales. Comparara este algoritmo en complejidad con la búsqueda binaria habitual.

Obsérvese que, mientras para la búsqueda binaria tenemos un orden exacto $\theta(\log n)$, para *busbin2* habremos probablemente de hacer un estudio en el peor caso, en el mejor y en media.

7. Diseñar un algoritmo de búsqueda ternaria *buster* que compara x (elemento a buscar) con el elemento de la posición $n/3$ y luego, si es necesario, con el de la posición $2n/3$, reduciendo en todo caso el número de posiciones a considerar a una tercera parte. Estudiar la complejidad de dicho algoritmo y compararlo con *busbin*. Recordar que se cuentan las casillas por examinar, y que por tanto, en *buster* se comprueba en ocasiones una casilla adicional (la de $2n/3$).

8. Sea T una tabla de n enteros parcialmente ordenados de modo que $T[1] \leq T[2] \leq ... \leq T[m]$ y $T[m+1] \leq T[m+2] \leq ... \leq T[n]$. ¿Es posible ordenar T en tiempo lineal usando una cantidad de espacio constante (no lineal en n, como una matriz R[1..n])?

9. Se tienen dos vectores A y B de n enteros que cumplen la propiedad de que son iguales componente a componente hasta una posición dada, y a partir de ella, son distintos componente a componente. Es decir, si A y B son iguales hasta la componente 7, eso significa que $A[i]=B[i]$ para $i=1,2,...,7$, y que $A[i] \neq B[i]$ para $i=8,9,...,n$. Por ejemplo: A=[2,3,8,4,5,6,9,1,4,7,4,9] y B=[2,3,8,4,5,6,9,6,3,5,1,8]. Escribir el pseudocódigo de un algoritmo que calcule cual es la primera posición en la que A y B son distintos (en el caso del ejemplo, la 8) y estudiar su complejidad. El algoritmo debe poseer una complejidad menor que la lineal (que, por ejemplo, obtendríamos recorriendo ambos vectores en paralelo).