Examen de Algorítmica y Complejidad (Plan 2014)

8 de noviembre de 2019

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema. Decimos que un *array*, v, de N enteros está ordenado circularmente si, o bien el vector está ordenado, o bien $v[N-1] \leq v[0]$ y $\exists k$ con $0<k<N$ tal que $\forall i \neq k$ $v[i] \leq v[i+1]$ (esto es, está ordenado imaginando que fuera un *array* circular).

Sea un *array* ordenado circularmente en el que todos los números se encuentran repetidos 2 veces salvo uno que aparece solo una vez. Se desea encontrar el elemento que aparece sólo una vez.

Ejemplo:

0	1	2	3	4	5	6	7	8
1	2	2	3	3	4	0	0	1

a) Diseñar un algoritmo basado en Divide y Vencerás con complejidad O(log N) en el caso peor[1] (donde N es el tamaño del vector) que devuelva el elemento que aparece una sola vez.

1ª Opción:

```
int elementoEspecial(int[] vector){
    if (vector.length==1)  return vector[0];
    else {
       int i0, iN;
       if (vector[0] == vector[vector.length-1]) { i0 = 1; iN = vector.length-2; }
       else {    i0 = 0;   iN = vector.length - 1;  }
        return elementoEspecialAux(vector,i0,iN);
    }
}

int elementoEspecialAux(int[] vector, int i0, int iN){
    if (i0==iN) return vector[i0];
    else {
       int k=(i0+iN)/2;
       if (vector[k]==vector[k+1]) {
          if ((k-1-i0+1) % 2 == 1) return elementoEspecialAux(vector, i0, k-1);
          else  return elementoEspecialAux(vector, k+2, iN);
       }
       else if (vector[k-1]==vector[k]) {
          if ((k-2-i0+1) % 2 == 1) return elementoEspecialAux(vector, i0, k-2);
          else  return elementoEspecialAux(vector, k+1, iN);
       }
       else  return vector[k];
    }
}
```

[1] Desarrollar un algoritmo que tenga una complejidad diferente a O(log *N*) en el caso peor conllevará una puntuación de 0 en la pregunta.

2ª Opción:

```
int elementoEspecial(int[] vector){
    return elementoEspecialAux(vector,0,vector.length-1);
}

int elementoEspecialAux(int[] vector, int i0, int iN){
    if (i0==iN)
        return vector[i0];
    else {
        int i0Aux, iNAux;
        if (vector[i0] == vector[iN]) { i0Aux = i0 + 1; iNAux = iN - 1; }
        else {    i0Aux=i0; iNAux=iN;  }
        int k=(i0Aux+iNAux)/2;
        if (vector[k]==vector[k+1]) {
          if ((k - 1 - i0Aux+1) % 2 == 1)
              return elementoEspecialAux(vector, i0Aux, k - 1);
          else
              return elementoEspecialAux(vector, k + 2, iNAux);
        }
        // aunque solo consideramos como caso base el tamaño 1, para los casos
        // generales el número de elementos entre i0 e iN es siempre impar, por lo
        // que podemos consultar la posición k-1 sin salirnos del rango i0..iN
        else if (vector[k-1]==vector[k]) {
            if ((k - 2 - i0Aux + 1) % 2 == 1)
                    return elementoEspecialAux(vector, i0Aux, k - 2);
                else
                    return elementoEspecialAux(vector, k + 1, iNAux);
        }
        else
            return vector[k];
    }
}
```

b) Justifica que la complejidad del algoritmo desarrollado en el apartado anterior para el caso peor es O(log *N*).

El algoritmo implementado obedece a la siguiente ecuación de recurrencia en el tiempo para N>1:

$$T(N) = T(N/2) + O(1)$$

Esta ecuación es del tipo $T(N) = p \cdot T(N/q) + f(N)$, donde $f(N) \in O(N^a)$, con p=1, q=2 y a=0, por lo que podemos aplicar el Teorema Maestro. Dado que $\log_q(p) = \log_2(1)=0$ y a=0 nos encontramos en el caso 2º del Teorema maestro ($a=\log_q(p)$), por lo que la complejidad del algoritmo es: $T(N) \in O\left(N^{\log_q(p)} \cdot \log N\right) = O(N^0 \log N) = O(\log N)$