LECCION 7: D y V

Ordenación: Quicksort

```
void Quicksort(int *v, int i, int j){
        if (i<j){
                int p= Pivote(v, i, j)
                Quicksort(v, i, p-1);
                Quicksort(v, p+1, j)
        }
}
int Pivote(int *v, int i, int j){
   int piv, k, l;
   piv = v[i];
   k = i;
   l = j+1 ;
   do{
      k = k+1;
   }while (v[k] ≤ piv && k<j);
   do{
      l = l-1;
   }while (v[l] > piv);
   while (k<l)
      Intercambiar(v[k], v[l]);
      do{
         k = k+1;
      }while (v[k] ≤ piv);
      do{
         l = l-1;
      }while (v[l] > piv);
   }
   Intercambiar(v[i], v[l]);
   return l;
}.
```

(1)

LECCION 7: D y V

Eficiencia Quicksort:

La eficiencia del Quicksort depende del pivote que se ha escogido. Si el pivote genera dos subvectores de igual tamaño (en promedio y mejor caso será así) el tiempo de ejecución se puede plantear de la siguiente forma:

- La función pivote hace en definitiva siempre un recorrido del vector luego cuesta siempre $\Theta(n)$. Por lo tanto en promedio Quicksort se puede plantear como:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n \geq 2 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$n = 2^m$$

$$T(2^m) - 2T(2^{m-1}) = 2^m \quad \begin{bmatrix} b=2 \\ p(m)=1 \\ d=0 \end{bmatrix}$$

$$x = T(2^m)$$

$$(x-2)(x-2) = 0$$

$$T(2^m) = c_1 \cdot 2^m + c_2 m \cdot 2^m$$

$$T(n) = c_1 \cdot n + c_2 \log_2(n) \cdot n$$

$$\in \Theta(\log_2(n) \cdot n)$$

El peor caso se da cuando el vector está ordenado.

El tiempo de ejecución de Quicksort ahora se plantea teniendo en cuenta que una de las mitades se queda con n-1 elementos y la otra 0.

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n \geq 2 \end{cases}$$

$$T(n) = T(n-1) + n$$

$$x = T(n)$$

$$(x-1)^3 = 0 \quad \begin{bmatrix} b=1 \\ p(n)=n \\ d=1 \end{bmatrix}$$

$$T(n) = c_1 1^m + c_2 n 1^n + c_3 n^2 1^n$$

$$\in \boxed{O(n^2)}$$

¿Si el vector está ordenado de forma creciente, cual es su T(n)?

②

LECCION 7: DyV

EJEMPLO: EL ELEMENTO EN SU POSICION

Sea $a[0 \ldots n-1]$ un vector ordenado de forma creciente y todos los elementos distintos. Tenemos que implementar un algoritmo de complejidad $O(\log n)$ en el peor caso capaz de encontrar un indice i tal que $0 \le i \le n-1$ y $a[i]=i$, suponiendo que existe

SOLUCION:

Como idea usar la busqueda binaria y teniendo en mente que si existe tal elemento debe ser uno de entre 0 y n-1 (para un vector de n posiciones).

Asi obtenemos la mitad del vector $a[n/2]$ y comprobamos si $a[\frac{n}{2}] = \frac{n}{2}$ si es asi lo hemos encontrado. Si no se puede dar 1) $a[\frac{n}{2}] > \frac{n}{2}$ en tal caso si existe el elemento debe estar entre 0 y $\frac{n}{2}-1$. Esto es asi porque el rango de valores menores que pueden haber a la derecha es

$$a[\tfrac{n}{2}]+1 > \tfrac{n}{2}+1$$

$$a[\tfrac{n}{2}]+2 > \tfrac{n}{2}+2$$

$$\vdots$$

$$a[\tfrac{n}{2}]+n-\tfrac{n}{2}-1 > \tfrac{n}{2}+n-\tfrac{n}{2}+1 = n-1$$

2) $a[\frac{n}{2}] < \frac{n}{2}$ el razonamieto es equivalente

```
int Posicion ( const int * v, int n){
  if (n==0)
     return -1;
  else{
     int mitad = n/2;
     if (v[mitad]== n/2) return mitad;
     if (v[mitad] > n/2)
     return Posicion(v, n/2)
     else
       return Posicion(v+n/2+1, n-n/2-1);
  }
}.
```

3

LECCIÓN 7.- DyV

Problema de la Selección.- Sea T un vector no ordenado con n posiciones. Y sea $\boxed{s}$ un entero entre 0...n-1. El problema de la selección consiste en encontrar el elemento que se encontraría en la posición s si el vector estuviera ordenado.

Ejemplo

	0	1	2	3	4	5	6	7	8	9 ← s
T	5	9	2	5	4	3	4	10	1	6

cuando T está ordenado

0	1	2	3	4	5	6	7	8	9 ← s
1	2	3	4	4	5	5	6	9	10

Si $s = \lceil \frac{n}{2} \rceil$ entonces nuestro problema es encontrar la mediana.

→ SOLUCION TRIVIAL ⟶ Ordenar el vector y obtener T[s]. Esta solución la podríamos obtener en $\Theta(n \log n)$.

. IDEA: Usar Pivote y reconducir la busqueda por la parte donde se encuentre s.

```
int Seleccion(int *v, int n, int s){
    int i,j,l;
    i=0; j=n-1;
    do{
        l = Pivote(v, i, j);
        if (s < l)
            j = l-1;
        else
            i = l+1;
    }while(l != s);
    return T[l];
}
```

- EFICIENCIA:
  - MEJOR CASO
    $T(n) = T(\frac{n}{2}) + n \in \Omega(n)$
  - PEOR CASO (array ordenado)
    $T(n) = T(n-1) + n \in O(n^2)$
  - CASO Promedio
    $T(n) = T(\frac{n}{2}) + n \in O(n)$

1.

LECCION 8. DyV

Ejemplo.- Dados n valores reales, diseñar un algoritmo que calcule el valor de la diferencia de entre los dos valores más cercanos.

$$d = \min_{\substack{0 \le i,j \le n-1 \\ i \ne j}} |v_i - v_j|$$

IDEA .- Usar el Quicksort como base. En el caso general obtenemos la distancia menor de la izquierda y la distancia menor de la parte derecha. Además la comparamos con la distancia del pivote al siguiente y anterior al pivote y con la distancia del pivote al siguiente y devolvemos la mínima. Caso base si tenemos un elemento devolvemos ∞ y si tenemos 2 la diferencia en valor absoluto.

EJEMPLO

5 7 1 1

Pivote

5 7 1 1

k

5 1 1 7

l k

$\underbrace{1 \quad 1}_{\text{caso base distancia } 0}$ $\boxed{5}$ $\underbrace{7}_{\text{Caso base distancia } \infty}$

distancia 4 — dista 2

min es 0.

```
float M_Distancia (const float * v, int n){
    if (n==2){
        return abs(v[0]-v[1]);
    }else if (n==1)
        return infinito;   //numeric_limits<float>::max();
    else{
        int p = Pivote ( v, 0, n-1);
        float i = M_Distancia( v, p);
        float d = M_Distancia (v+p+1, n-p-1);
        float m_1 = min(i, d);
        float m_2 = min (abs(v[p-1],v[p]), abs [v[p],v[p+1]));
        return ((m_1 > m_2) ? m_2 : m_1);
}
```