Programación2

Anexo a la lección 16.
Análisis del coste del algoritmo quicksort de ordenación de vectores

Problema. Caracterizar asintóticamente el coste, en tiempo, de la invocación `quicksort(v,n)`.

```
// Pre: v = Vo  AND  n > 0  AND  n <= #v
// Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
template <typename T>
void quicksort (T v[], const int n) {
   //  Ordenación del vector v[0,n-1] aplicando el método
   //  de ordenación rápida de Hoare o quicksort
   quicksort(v, 0, n-1);
}
```

Donde:

```
esPermutación(v1,v2,desde,hasta) =
     (PT alfa EN [desde,hasta].
         (Núm beta EN [desde,hasta]. v1[beta] = v1[alfa]) =
         (Núm beta EN [desde,hasta].v2[beta] = v1[alfa]) )
ordenado(v,desde,hasta) = (PT alfa EN [desde,hasta-1]. v[alfa]<=v[alfa+1])
```

```
/*
 * Pre: v = Va  AND  izda >= 0 AND  dcha < #v
 * Post: esPermutación(v,Va,izda,dcha) AND ordenado(v,izda,dcha)
 */
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
   if (izda < dcha) {
      T pivote = v[izda];
      int i = izda + 1,  d = dcha;
      while (i != d+1) {
         if (v[i] <= pivote) { i = i + 1; }
         else if (v[d] >= pivote)  { d = d - 1; }
         else {
            T dato = v[i];
            v[i] = v[d]; v[d] = dato; i = i + 1; d = d - 1;
         }
      }
      v[izda] = v[d]; v[d] = pivote;
      quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);
   }
}
```

Caracterizar asintóticamente el coste, en tiempo, de la invocación quicksort(v,n).

$$t_{quicksort(v,n)}(n) = t_{inv.} + t_a + t_{quicksort(v,0,n-1)}(n)$$

```
// Pre: v = Vo  AND  n > 0  AND  n <= #v
// Post: esPermutación(v,Vo,0,n-1) AND ordenado(v,0,n-1)
template <typename T>
void quicksort (T v[], const int n) {
    // Ordenación del vector v[0,n-1] aplicando el método
    // de ordenación rápida de Hoare o quicksort
    quicksort(v, 0, n-1);                          // a
}
```

```
// Pre: …       Post: …
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
   if (izda < dcha) {                                          // a
      T pivote = v[izda];                                      // b
      int i = izda + 1, d = dcha;                              // c
      while (i != d + 1) {                                     // d
         if (v[i] <= pivote)                                   // e
         { i = i + 1; }                                        // f
         else if (v[d] >= pivote)                              // g
         { d = d - 1; }                                        // h
         else {
            T dato = v[i]; v[i] = v[d]; v[d] = dato;           // i
            i = i + 1;  d = d - 1;                             // j
         }
      }
      v[izda] = v[d]; v[d] = pivote;                           // k
      quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);       // l
   }
}
```

```
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
   if (izda<dcha) {                                                    // a
      T pivote = v[izda];                                              // b
      int i = izda + 1, d = dcha;                                      // c
      while (i != d + 1) {                                             // d
         if (v[i] <= pivote)                                           // e
         { i = i + 1; }                                                // f
         else if (v[d] >= pivote)                                      // g
         { d = d - 1; }                                                // h
         else {
            T dato = v[i]; v[i] = v[d]; v[d] = dato;                   // i
            i = i + 1;  d = d - 1;                                     // j
         }
      }
      v[izda] = v[d]; v[d] = pivote;                                   // k
      quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);               // l
   }
}
```

n = dcha – izda + 1

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + t_{while} + t_k + t_l + t_{quicksort(v,izda,d-1)} + t_{quicksort(v,d+1,dcha)}$$

```
int i = izda + 1, d = dcha;                                  // c
while (i != d + 1) {                                         // d
   if (v[i] <= pivote)                                       // e
   { i = i + 1; }                                            // f
   else if (v[d] >= pivote)                                  // g
   { d = d - 1; }                                            // h
   else {
      T dato = T[i]; T[i] = T[d]; T[d] = dato;               // i
      i = i + 1;  d = d - 1;                                 // j
   }
}
```

n = dcha – izda + 1

Cálculo de $\mathbf{t_{while}(n)}$ (análisis de casos extremos):

- Caso en que siempre se satisfaga: **v[i] <= pivote**
- Caso en que siempre se satisfaga: **¬(v[i] <= pivote) ∧ (v[d] >= pivote)**
- Caso en que siempre se satisfaga: **¬(v[i] <= pivote) ∧ ¬(v[d] >= pivote)**

Cálculo de $t_{while}(n)$ (análisis de casos extremos):

- Caso en que siempre se satisfaga `v[i] <= pivote`

  $t_{while}(n) = (t_e + t_f + t_d).(n-1)$

- Caso en que siempre se satisfaga $\neg$`(v[i] <= pivote)` $\wedge$ `(v[d] >= pivote)`

  $t_{while}(n) = (t_e + t_g + t_h + t_d).(n-1)$

- Caso en que siempre se satisfaga $\neg$`(v[i] <= pivote)` $\wedge$ $\neg$`(v[d] >= pivote)`

  $t_{while}(n) = (t_e + t_g + t_i + t_j + t_d).(n-1) / 2$

Caracterización asintótica de $t_{while}(n)$ en estos casos extremos:

- En cualquiera de los casos posibles:

  $O(t_{while}(n)) = O(n)$

- Por lo tanto, es válida la siguiente aproximación asintótica:

  $t_{while}(n) = k.n$ (donde k es una constante real)

```
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
   if (izda < dcha) {                                          // a
      T pivote = v[izda];                                      // b
      int i = izda + 1, d = dcha;                              // c
      while (i != d + 1) {                                     // d
         …
      }
      v[izda] = v[d]; v[d] = pivote;                           // k
      quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);       // l
   }
}
```

n = dcha – izda + 1

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t_{quicksort(v,izda,d-1)} + t_{quicksort(v,d+1,dcha)}$$

¿Cabe distinguir **casos mejores** y **peores** en cuanto a coste?

¿**Cuándo** se presenta cada caso?

n = dcha – izda + 1

```
template <typename T>
void quicksort (T v[], const int izda, const int dcha) {
   if (izda < dcha) {                                          // a
      T pivote = v[izda];                                      // b
      int i = izda + 1, d = dcha;                              // c
      while (i != d + 1) {                                     // d
         …
      }
      v[izda] = v[d]; v[d] = pivote;                           // k
      quicksort(v, izda, d-1);  quicksort(v, d+1, dcha);       // l
   }
}
```

$$t(n) = t_{invoc.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t_{quicksort(v,izda,d-1)} + t_{quicksort(v,d+1,dcha)}$$

Casos mejores: La elección del pivote es afortunada y al final **d ≈(dcha+izda)/2**

Casos peores: La elección del pivote no es afortunada y al final **d ≈izda** ∨ **d ≈dcha**

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t_{quicksort(v,izda,d-1)} + t_{quicksort(v,d+1,dcha)}$$

n = dcha – izda + 1

Casos mejores: La elección del pivote es afortunada y al final **d ≈(dcha+izda)/2**

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t(n/2) + t(n/2)$$

$$t(n) - 2.t(n/2) = t_{inv.} + t_a + t_b + t_c + t_d + t_k + t_l + k.n$$

[La resolveremos un poco más adelante]

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t_{quicksort(v,izda,d-1)} + t_{quicksort(v,d+1,dcha)}$$

$$\mathbf{n = dcha - izda + 1}$$

Casos peores: La elección del pivote no es afortunada y

al final $\mathbf{d \approx izda \vee d \approx dcha}$

$$t(n) = t_{inv.} + t_a + t_b + t_c + t_d + \mathbf{k.n} + t_k + t_l + t(n-1) + t(0)$$

$$t(n) - t(n-1) = t_{inv.} + t_a + t_b + t_c + t_d + t_k + t_l + t(0) + k.n$$

Ecuación característica:

$$(x-1)(x-1)^2 = 0$$

Solución:

$$t(n) = c_1.n^2.1^n + c_2.n.1^n + c_3.1^n = c_1.n^2 + c_2.n + c_3$$

Caracterización asintótica del caso peor:

$$\boldsymbol{O(t(n))} = \boldsymbol{O}(c_1.n^2 + c_2.n + c_3) = \boldsymbol{O(n^2)}$$

Casos mejores: La elección del pivote es afortunada y al final **d ≈(dcha+izda)/2**

$$t(n) - 2.t(n/2) = t_{inv.} + t_a + t_b + t_c + t_d + t_k + t_l + k.n$$

Cambio de variable $m = \log_2 n$ [$n = 2^m$ y $\log_2(n/2) = m-1$]:

$$t(m) - 2.t(m-1) = t_{inv.} + t_a + t_b + t_c + t_d + t_k + t_l + k.2^m$$

Despreciamos los términos constantes $t_a + t_b + t_c + t_d + t_k + t_l$ ante el término $k.2^m$, para poder así continuar con el cálculo:

$$t(m) - 2.t(m-1) = k.2^m$$

Ecuación característica:

$(x-2)(x-2) = 0 \quad \rightarrow \quad$ Raíces: $x = 2$ (doble)

Solución:

$$t(m) = c_1.m.2^m + c_2.2^m \rightarrow t(n) = c_1.n.\log_2 n + c_2.n$$

Caracterización asintótica del caso mejor:

$$\boldsymbol{O(t(n))} = \boldsymbol{O}(c_1.n.\log_2 n + c_2.n) = \boldsymbol{O(n.\log n)}$$

Volviendo al problema inicial de caracterizar asintóticamente el coste de la invocación **`quicksort(v,n)`**.

$$t_{quicksort(v,n)} = t_{inv.} + t_a + t_{quicksort(v,0,n-1)}$$

En los **casos mejores** :

$$O(t_{quicksort(v,0,n-1)}) = O(n.\log n)$$

Por lo tanto, en estos casos:

$$O(t_{quicksort(v,n)}) = O(t_a + t_{quicksort(v,0,n-1)}) = O(t_a + k.n.\log n) = O(n.\log n)$$

En los **casos peores**:

$$O(t_{quicksort(v,0,n-1)}) = O(n^2)$$

Por lo tanto, en estos casos:

$$O(t_{quicksort(v,n)}) = O(t_{inv.} + t_a + t_{quicksort(v,0,n-1)}) = O(t_{inv.} + t_a + k.n^2) = O(n^2)$$