PRG - ETSInf. TEORIA. Curs 2013-14. Parcial 1.
14 d'abril de 2014. Duració: 2 hores.

1. [3 punts] Donat un array `a` de `int`, escriure un mètode **recursiu** que sume els números capicua des dels extrems de l'array, definits per l'interval [`ini`,`fi`] (0≤`ini`,`fi`<`a.length`), cap al centre fins sumar-los tots o trobar algun parell de números que no siguen iguals. En cas que l'array tinga una quantitat senar d'elements, l'element central de l'interval només ha de sumar-se una vegada. Si l'array no té cap element capicua en els extrems de l'interval, el resultat és zero.

   Alguns exemples:

   - per a l'array `a = {1,4,1,2,4,1}`, el resultat de la suma és 10 (1+1+4+4),
   - per a l'array `a = {1,4,2,2,4,1}`, el resultat de la suma és 14 (1+1+4+4+2+2),
   - per a l'array `a = {1,4,1,4,1}`, el resultat de la suma és 11 (1+1+4+4+1),
   - per a l'array `a = {1,4,0,0,1}`, el resultat de la suma és 2 (1+1),
   - per a l'array `a = {8,4,0,0,1}`, el resultat de la suma és 0.

   Es demana:

   a) (0.5 punts) Perfil del mètode, afegint el/els paràmetres adequats per tal de resoldre recursivament el problema.

   Solució: Una possible solució consisteix en definir un mètode amb el següent perfil:

   ```
   public static int sumaCapicua(int[] a, int ini, int fi)
   ```

   sent 0≤`ini` i `fi`<`a.length`.

   b) (1.2 punts) Cas base i cas general.

   Solució:

   - Cas base, `ini>fi`: Subarray buit. Retorna el neutre de la suma 0.
   - Cas base, `ini==fi`: Subarray d'un element. Retorna `a[ini]`.
   - Cas general, `ini<fi`: Si els extrems no són capicua (`a[ini]!=a[fi]`), retorna el neutre de la suma 0. En un altre cas (`a[ini]==a[fi]`), el resultat és la suma dels valors dels extrems `a[ini]` i `a[fi]` més la suma dels valors capicua des dels extrems del subarray `a[ini+1..fi-1]`.

   c) (1 punt) Implementació en Java.

   Solució:

   ```
   /** Suma dels valors capicua des dels extrems del subarray
    *  definit per les posicions ini i fi, 0<=ini, fi<a.length.
    *  Si el subarray està buit o no existeixen capicua en els extrems,
    *  el resultat és zero.
    */
   public static int sumaCapicua(int[] a, int ini, int fi) {
   ```

```
    if (ini>fi) return 0;
    else if(ini==fi) return a[ini];
    else if (a[ini]!=a[fi]) return 0;
    else return a[ini] + a[fi] + sumaCapicua(a, ini+1, fi-1);
}
```

d) (0.3 punts) Crida inicial.

Solució: Per a un array `a`, la crida `sumaCapicua(a,0,a.length-1)` resol el problema de l'enunciat.

2. 3 punts Considerar el següent mètode recursiu en Java que comprova si tots els elements del subarray `a[pos..a.length-1]` apareixen formant una progressió aritmètica de diferència `d`:

```
/** Retorna true si per a tota parella a[i],a[i+1] en a[pos..a.length-1]
 *  es compleix que a[i+1] = a[i]+d, i false en cas contrari.
 *  Precondició:
 *  a.length>=1 && 0<=pos<=a.length-1
 */
public static boolean progAritmetica(int[] a, int d, int pos) {
    if (pos==a.length-1) return true;
    else return (a[pos+1]==a[pos]+d) && progAritmetica(a, d, pos+1);
}
```

Es demana:

a) (0.25 punts) Indicar quina és la talla del problema i quina expressió la defineix.

Solució: La talla és el nombre d'elements de l'array en consideració en cada crida. L'expressió que la defineix és `a.length-pos`, que anomenarem n.

b) (0.5 punts) Identificar, cas de que les haguera, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

Solució: Existeixen diferents instàncies per què és un problema de cerca en un array.

El *cas millor* es presenta quan la condició no es compleix per al primer parell d'elements considerats, és a dir, quan la diferència entre les dues primeres components del subarray considerat no és `d` (açò és, `a[pos+1]`$\neq$`a[pos]+d`), de manera que torna `false` sense fer cap crida recursiva.

El *cas pitjor* ocorre quan la condició es compleix per a tots els parells d'elements, és a dir, quan la diferència entre tots els parells d'elements consecutius del subarray considerat és `d`, de manera que torna `true` quan s'arriba al cas base, després de realitzar el número màxim de crides recursives.

c) (1.5 punts) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Resoldre-la(les) per substitució.

Solució:
En el cas millor, per a qualsevol talla es té: $T^m(n) = k$, sent k una constant positiva, en alguna unitat de temps.
En el cas pitjor, el cost s'expressa recurrentment com:

$$T^p(n) = \begin{cases} k_0 & si\ n = 1 \\ k_1 + T^p(n-1) & si\ n > 1 \end{cases}$$

sent k_0, k_1 constants positives, en alguna unitat de temps. Resolent per substitució:

$$\begin{aligned} T^p(n) &= k_1 + T^p(n-1) = \ 2 \cdot k_1 + T^p(n-2) = \ 3 \cdot k_1 + T^p(n-3) = \cdots = \\ &= i \cdot k_1 + T^p(n-i) = \cdots = \\ & (cas\ base:\ n - i = 1,\ i = n-1) \\ &= k_1 \cdot (n-1) + T^p(1) = k_1 \cdot n + (k_0 - k_1) \end{aligned}$$

d) (0.75 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

$$T^m(n) \in \theta(1),\ T^p(n) \in \theta(n).$$

$$T(n) \in \Omega(1),\ T(n) \in O(n).$$

3. $\boxed{\text{4 punts}}$ El següent mètode, `triangle(int)`, determina, escrivint-los, el número de triangles rectangles de costats enters i hipotenusa `h`:

```
/** El mètode compta, escrivint-los, tots els triangles
 *  rectangles de costats enters i hipotenusa h. */
public static int triangle(int h) {
   int cont = 0;
   for (int c1 = 4; c1 < h; c1++)
      for (int c2 = 3; c2 < c1; c2++)
           if (c1*c1 + c2*c2 == h*h) {
              cont++;
              System.out.println("c1= " + c1 + ", c2= " + c2 + ", h= " + h);
           }
   return cont;
}
```

Es demana:

a) (0.5 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

Solució: La talla n del problema és el paràmetre `h`, és a dir, la hipotenusa dels triangles buscats.

b) (0.5 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

Solució: No hi ha diferents instàncies, el número de passades que fan els bucles només depèn de la talla.

c) (2 punts) Triar una unitat de mesura per al cost (passos de programa, instrucció crítica) i d'acord amb ella, obtenir una expressió el més precisa possible del cost temporal del programa, a nivell global o en les instàncies més significatives si és el cas.

Solució:
En passos de programa:

$$T(n) = 1 + \sum_{i=4}^{n-1}(1 + \sum_{j=3}^{i-1} 1) = 1 + \sum_{i=4}^{n-1}(i-2) = 1 + \frac{(n-1)\cdot(n-4)}{2} \ p.p.$$

Si triem com instrucció crítica l'avaluació de la condició `c1*c1 + c2*c2 == h*h`, el que equival a despreciar termes d'ordre inferior:

$$T(n) = \sum_{i=4}^{n-1}\sum_{j=3}^{i-1} 1 = \sum_{i=4}^{n-1}(i-3) = \frac{(n-3)\cdot(n-4)}{2} \ \textit{instruccions crítiques}$$

d) (1 punt) Expressar el resultat anterior en notació asimptòtica.

Solució: $T(n) \in \theta(n^2)$.