PRG - ETSInf. TEORIA. Curs 2012-13. Parcial 1.
22 d'abril de 2013. Duració: 2 hores.

1. 4 punts Donat un array `a` d'`int` i un enter `m`, escriure un mètode **recursiu** que comprove si existeix una parella `a[i]` i `a[f]`, $0 \leq$`i`$\leq$`f`<`a.length`, de components *simètriques* (la distància d'`i` a 0 és la mateixa que la distància de `f` a `a.length`-1) que sumen `m`. Si existeix la parella, ha de tornar l'índex en el que es troba (l'índex `i`, el més baix de la parella), -1 en cas contrari.

   Per exemple, per a `m`=4 i `a`={1,4,5,9,6,0,-8} el mètode en la crida inicial que s'extenga sobre tot l'array ha de tornar 1, per a `m`=18 i `a` ={1,4,5,9,6,0,2} ha de tornar 3, per a `m`=25 i `a`={1,3,2,5,4,6} ha de tornar -1.

   S'ha d'indicar:

   a) Perfil del mètode que es va escriure, afegint el/s paràmetre/s adequat/s per a resoldre recursivament el problema.

   Solució: Una possible solució consisteix a definir el mètode com

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

   de manera que éssent $0\leq$`ini`, `fi<a.length`, es delimita a cercar una parella d'elements simètrics en el subarray `a[ini..fi]`.

   b) Cas base i cas general.

   Solució:

   - Cas base `ini>fi`: No es troba la parella buscada, s'ha de retornar -1.
   - Cas general, `ini<=fi`. Si `a[ini]+a[fi]` val `m`, s'ha de retornar `ini`, sino la cerca es redueix a `a[ini+1..fi-1]`.

   c) Implementació en Java.

   Solució:

   ```
   /** Busca l'índex de la primera parella d'elements simètrics en a[ini..fi],
     * 0<=ini, fi<a.length, que sumats dóna m. Si no existeix, retorna -1.
     */
     public static int parellaSim(int[] a,int ini,int fi,int m){
        if (ini>fi) return -1;
        else if (a[ini]+a[fi]==m) return ini;
        else return parellaSim(a,ini+1,fi-1,m);
     }
   ```

   d) Crida inicial.

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

2. $\boxed{\text{3 punts}}$ Donada certa matriu `m` quadrada i un array `a`, d'enters, els dos amb la mateixa dimensió (això és, `m.length==a.length`), el següent mètode **iteratiu** torna la posició (número de fila) on es troba l'array `a` en `m`, cas de que es trobe, o torna -1 si no fora així:

```
/**
 * Torna la posició en que l'array "a" es troba com a fila dins de la matriu
 * quadrada "m", o -1 si no es troba.
 * PRECONDICIO: m es quadrada i m.length == a.length
 */
public static int cercaPosFila(int[][] m, int[] a) {
    boolean trobat = false;
    int i = 0;
    while (i<m.length && !trobat) {
        trobat = true;
        for (int j=0; j<m.length && trobat; j++)
            trobat = (m[i][j]==a[j]);
        if (!trobat) i++;
      }
    if (trobat) return i; else return -1;
}
```

Es demana l'estudi del seu cost temporal:

a) Indica quina és la grandària o talla del problema, així com l'expressió que la representa.

> **Solució:** La talla o grandària del problema és $n =$ `m.length`, és a dir, la dimensió de la matriu.

b) Identifica, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

> **Solució:** El mètode és un problema de cerca i, per tant, per a una mateixa talla sí que presenta instàncies distintes.
>
> *Cas millor*: Per una banda, si les n components d'`a` apareixen en la primera fila de la matriu, el bucle exterior sols executa una passada, en la que es comprova en n passades que les `n` successives components d'`a[0..n-1]` coincideixen amb les corresponents components de `m[0]`.
>
> Per altra banda, es pot donar que el bucle intern siga lo més curt possible per a totes les files si el primer element de cada fila és diferent d'`a[0]`. En eixe cas el bucle exterior completaria les n passades (una per cada fila de la matriu), i totes elles amb el cost de fer una sola comprovació.
>
> *Cas pitjor*: Ocorre quan per a totes les files es fan el màxim de comprovacions possibles, és a dir, les components d'`a[0..n-2]` coincideixen amb les corresponents de la fila, però `a` no apareix complet en cap fila de `m` (excepte potser la darrera).

c) Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obté una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.

> **Solució:**
>
> - Si optem per triar com unitat de mesura el pas de programa, s'obté:
>
>   En el cas millor, $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ passos.

En el cas pitjor, $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1 + n) = 1 + n + n^2$ passos.

- Si optem per triar la instrucció crítica com unitat de mesura per a l'estimació del cost, es pot considerar com a tal la comparació: `(m[i][j] == a[j])`.

  En el *cas millor* s'executarà n vegades i la funció de cost temporal en aquest cas serà $T^m(n) = n$.

  En el *cas pitjor* es repetirà el número màxim de vegades possible i la funció de cost serà $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n \cdot n = n^2$.

d) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució: $T(n) \in \Omega(n)$, $T(n) \in O(n^2)$.

3. $\boxed{\text{2 punts}}$ El següent mètode comprova si cert subarray `a[i..f]` d'`int` està o no ordenat ascendentment subdividint-lo prèviament i comprobant que cadascuna de les seues dues parts ho estiga, així com la relació entre elles:

```
/** Torna cert sii a[i..f] està ordenat ascendentment. Precondició: i<=f */
 public static boolean esOrdenat(int[] a, int i, int f) {
     if (i==f) return true;
     else {
         int m = (i+f)/2;
         return esOrdenat(a,i,m) && esOrdenat(a,m+1,f) && a[m]<=a[m+1];
     }
 }
```

Es demana l'estudi del seu cost temporal:

a) Indica quina és la talla del problema i quina expressió la defineix.

Solució: La talla, n, és l'amplària del subarray comprès entre les posicions i i f. Això és, $n = f - i + 1$.

b) Determina si existeixen instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

Solució: El cas millor és aquell en el que sempre s'executa només la primera crida, això és, quan el primer element està fora d'ordre (quan inicialment `a[i]` $>$ `a[i+1]`). El cas pitjor és aquell en el qual s'han d'executar totes les crides recursives. Aquesta situació es dóna, per exemple, quan inicialment el subarray `a[i..f]` està ordenat.

c) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

Solució:

Cas millor:

$$T^m(n) = \begin{cases} k & n = 1 \\ T^m(\frac{n}{2}) + k' & n > 1 \end{cases}$$

$$\begin{array}{lll} 1) & T^m(n) = & T^m(\frac{n}{2}) + k' \\ 2) & & T^m(\frac{n}{2^2}) + 2k' \\ 3) & & T^m(\frac{n}{2^3}) + 3k' \\ .. & & ... \\ i) & & T^m(\frac{n}{2^i}) + ik' \end{array}$$

$\frac{n}{2^i} = 1; \quad n = 2^i; \quad log_2 n = i;$

$T^m(n) = k + k' log_2 n$

Cas pitjor:

$$T^p(n) = \begin{cases} k & n = 1 \\ 2T^p(\frac{n}{2}) + k' & n > 1 \end{cases}$$

$$\begin{array}{lll} 1) & T^p(n) = & 2T^p(\frac{n}{2}) + k' \\ 2) & & 2^2 T^p(\frac{n}{2^2}) + k'(1+2) \\ 3) & & 2^3 T^p(\frac{n}{2^3}) + k'(1+2+2^2) \\ .. & & ... \\ i) & & 2^i T^p(\frac{n}{2^i}) + k' \sum_{j=0}^{i-1} 2^j \end{array}$$

$\frac{n}{2^i} = 1; \quad n = 2^i;$

$T^p(n) = 2^i k + (2^i - 1)k' = nk + (n-1)k' = n(k + k') - k'$

d) Expressa el resultat anterior fent servir notació asimptòtica.

Solució:

$$T(n) \in \Omega(log\ n)$$
$$T(n) \in O(n)$$

4. $\boxed{\text{1 punt}}$ És possible modificar una única instrucció de l'algorisme anterior per a que el seu cost temporal siga $\Omega(1)$.

Quina instrucció s'hauria de modificar i de quina forma?

Solució: Cal modificar l'instrucció amb les crides recursives per a que s'execute primer la comprovació, això és, caldria substituir-la per:

```
return a[m]<=a[m+1] && esOrdenat(a,i,m) && esOrdenat(a,m+1,f);
```