# PRG - ETSInf. TEORIA. Curs 2016-17. Recuperació Parcial 1.
# 26 de juny de 2017. Duració: 2 hores.

**Nota:** L'examen s'avalua sobre 10 punts però el seu pes específic en la nota final de PRG és de **3 punts**.

1. [3 punts] Un número triangular és aquell que pot recomposar-se en la forma d'un triangle equilàter (considerarem que el primer número triangular és el 0). Els primers 10 números triangulars són: 0, 1, 3, 6, 10, 15, 21, 28, 36, 45.

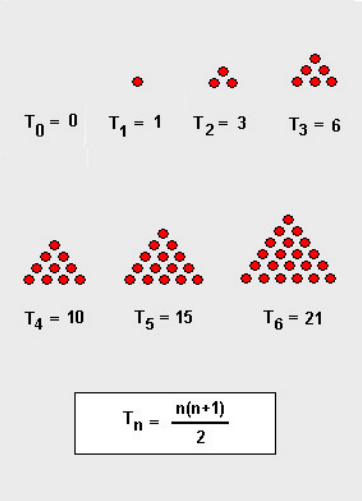


Figura 1: Representació gràfica dels 7 primers números triangulars

Cada número triangular es pot calcular com:

$$t(n) = \begin{cases} 0 & n = 0 \\ n + t(n-1) & n > 0 \end{cases}$$

On $t(n)$ és l'enèsim número triangular i $t(n-1)$ el seu anterior.

Es vol implementar, usant recursió, un algorisme que permeta instanciar un array d'un cert tamany $n$, i que continga els $n$ primers termes d'aquesta successió dels números triangulars.

**Es demana** implementar els següents mètodes, per als que a més s'hauran d'especificar les precondicions necessàries:

- Un primer mètode, llançadora, no recursiu, que reba un únic argument, el número $n$, que haurà de ser un valor enter positiu. Aquest mètode haurà d'instanciar l'array al tamany adequat, invocar al segon mètode (encarregat de reomplir l'array), i retornar l'array.
- Un segon mètode, recursiu, que reba dos arguments: l' array de tamany $n$ i un altre argument, necessari per a implementar el disseny recursiu. En aquest mètode s'hauran de calcular els valors de la successió, i magatzemmar-los en les posicions que corresponga en l'array.

Per exemple, la crida al mètode llançadora amb $n = 5$ haurà de retornar l'array $\{0, 1, 3, 6, 10\}$, mentre que amb $n = 1$ haurà de retornar l'array $\{0\}$.

**Solució:** Una possible solució consisteix a calcular cada nombre triangular, en el cas general, mitjançant l'equació $t(n) = n + t(n-1)$.

```
/** Precondició: n>0 */
public static int[] creaSuccessioTriangulars(int n) {
    int[] a = new int[n];
    creaSuccessioTriangulars(a, n - 1);
    return a;
}
```

```
/** Precondició: 0 <= n < a.length */
private static void creaSuccessioTriangulars(int[] a, int n) {
    if (n == 0) { a[n] = 0; }
    else {
        creaSuccessioTriangulars(a, n - 1);
        a[n] = n + a[n - 1];
    }
}
```

Una altra possible solució consisteix a calcular cada nombre triangular mitjançant l'equació $t(n) = n * (n + 1)/2$ mostrada en la Figura 1. En aquest cas es presenta, a més, una solució recursiva basada en descomposició ascendent.

```
/** Precondició: n > 0 */
public static int[] creaSuccessioTriangulars(int n) {
    int[] a = new int[n];
    creaSuccessioTriangulars(a, 0);
    return a;
}

/** Precondició: 0 <= n < a.length */
private static void creaSuccessioTriangulars(int[] a, int n) {
    if (n < a.length) {
        a[n] = n * (n + 1) / 2;
        creaSuccessioTriangulars(a, n + 1);
    }
}
```

2. 4 punts El següent mètode, donat un array `a` ordenat ascendentment, torna el número de vegades que apareix a l'array el primer número repetit (el de valor més menut) o 0 si no hi ha repetits a l'array. Per exemple, si `a` = {2, 5, 5, 5, 8, 8, 9} torna 3 (el primer número que apareix repetit és el 5 i es repeteix 3 vegades); si `a` = {2, 5, 8, 9} torna 0.

```
/** Precondició: a ordenat ascendentment */
public static int comptarPrimerRepetit(int[] a) {
    int compt = 0, i = 0;
    boolean repe = false;
    while (i < a.length - 1 && !repe) {
        int j = i + 1;
        repe = (a[j] == a[i]);
        while (j < a.length && a[j] == a[i]) { j++; }
        if (repe) { compt = j - i; }
        i++;
    }
    return compt;
}
```

**Es demana:**

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

e) (1 punt) Tenint en compte que l'algorisme d'ordenació per *inserció directa* vist en classe té un cost lineal en el millor cas i quadràtic en el pitjor, quin és el cost d'ordenar `a` per inserció directa i després aplicar el mètode `comptarPrimerRepetit(int[])`?

**Solució:**

a) La talla del problema és el nombre d'elements de l'array `a` i l'expressió que la representa és `a.length`. D'ara endavant, anomenarem a aquest número $n$. Açò és, $n =$ `a.length`.

b) Sí que existeixen diferents instàncies ja que es tracta d'una cerca del primer valor repetit que conté un altra cerca per tal de comptar quantes vegades es repeteix aquest valor.

El cas millor es dóna quan `a[0]` està repetit en `a[1]`, és a dir, `a[0] == a[1]`.

El cas pitjor es dóna quan no hi ha elements repetits a l'array i també quan `a[0]` està repetit en tot l'array, és a dir, $\forall$`i:1 ≤ i <a.length: a[0] == a[i]`.

c) Triant com a unitat de mesura el pas de programa, es té:

- Cas millor: $T^m(n) = 3$ *p.p.*
- Cas pitjor: $T^p(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*

Triant com a unitat de mesura la instrucció crítica i considerant com tal la guarda del bucle intern `j < a.length && a[j] == a[i]` (de cost unitari), es té:

- Cas millor: $T^m(n) = 2$ *i.c.*
- Cas pitjor: $T^p(n) = \sum_{i=0}^{n-2} 1 = n - 1$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.

e) La resolució del problema aplicant primer inserció directa i després el mètode `comptarPrimerRepetit` tindrà les següents fites de complexitat:

- En el cas millor, l'array està ordenat ascendentment i `a[0]` està repetit en `a[1]`. Aplicar l'ordenació per inserció directa tindrà un cost $\Theta(n)$ i aplicar el mètode `comptarPrimerRepetit` serà $\Theta(1)$. Per tant, el cost en el cas millor serà lineal amb la talla del problema.
- En el cas pitjor, els elements de l'array estan ordenats descendentment i no hi ha repetits. El cost de l'ordenació per inserció directa serà $\Theta(n^2)$ i aplicar el mètode `comptarPrimerRepetit` serà $\Theta(n)$. Per tant, el cost en el cas pitjor serà quadràtic amb la talla del problema.

3. [3 punts] El següent mètode calcula la suma dels bits de la representació en binari d'un enter no negatiu `num`.

```
/** Precondició: num >= 0 */
public static int sumaBits(int num) {
    if (num <= 1) { return num; }
    else {
        int ultBit = num % 2;
        return sumaBits(num / 2) + ultBit;
    }
}
```

**Es demana:**

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

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

c) (1.5 punts) Escriure 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.

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

**Solució:**

a) La talla del problema és `num`. D'ara endavant, anomenarem a aquest número $n$. Açò és, $n =$ `num`.

b) No existeixen instàncies significatives; per a una mateixa talla $n$, el mètode sempre realitza el mateix número d'operacions.

c) L'equació de recurrència es defineix com:

$$T(n) = \begin{cases} T(n/2) + 1 & \text{si } n > 1 \\ 1 & \text{si } n \leq 1 \end{cases}$$

Resolent per substitució:

$T(n) = T(n/2) + 1 = T(n/4) + 2 = T(n/8) + 3 = \ldots = T(n/2^i) + i$.

Si $n/2^i = 1 \rightarrow i = log_2 n$, amb el que $T(n) = T(n/2^{log_2 n}) + log_2 n = T(1) + log_2 n = 1 + log_2 n$ *p.p.*

d) En notació asimptòtica: $T(n) \in \Theta(log_2 n)$

Una solució alternativa consisteix a considerar com talla el número de xifres de la representació en base 2 del valor `num`. En aquest cas:

a) $m =$ número de xifres de la representació en base 2 de `num`.

b) No existeixen instàncies significatives; per a una mateixa talla $m$, el mètode sempre realitza el mateix número d'operacions.

c) L'equació de recurrència es defineix com:

$$T(m) = \begin{cases} T(m-1) + 1 & \text{si } m > 1 \\ 1 & \text{si } m \leq 1 \end{cases}$$

Resolent per substitució:

$T(m) = T(m-1) + 1 = T(m-2) + 2 = T(m-3) + 3 = \ldots = T(m-i) + i$.

Si $m - i = 1 \rightarrow i = m - 1$, amb el que $T(m) = T(1) + m - 1 = 1 + m - 1 = m$ *p.p.*

d) En notació asimptòtica: $T(m) \in \Theta(m)$

Com es pot observar, el cost temporal del mètode és el mateix, encara que expressat en funció de talles diferents. Recorda que el nombre de xifres de la representació en base 2 d'un número enter $n$ és $1 + \lfloor log_2 n \rfloor$, açò és, $m$.