Primer examen parcial de PRO2 **Durada: 2h 40m** **7/05/2018**

Cognoms **Nom** **DNI**

Problema 1 (5 punts)

1.1 Implementeu **iterativa i eficientment** l'acció `fusio_ord(t1,t2)`, que fusiona els elements de dues llistes ordenades `t1` i `t2`. Per exemple, si `t1 = {1,3,6,10}` i `t2 = {-2,0,5}` abans de la crida `fusio_ord(t1,t2)`, després d'aquesta els seus valors seran `t1 = {-2,0,1,3,5,6,10}` i `t2 = {}`. **(2 punts)**

void *fusio_ord* (*list* <**int**>& *t1*, *list* <**int**>& *t2*) {
/* Pre: t1 = T1, t2 = T2, T1 i T2 estan ordenades en ordre creixent. */
/* Post: t1 conté els elements de T1 i de T2 ordenats en ordre creixent. t2 és buida. */
/* Cost: lineal en la suma de les mides de t1 i t2. */

1.2 Escriviu l'invariant del primer o únic bucle que useu en la vostra implementació. Hauria de ser prou complet perquè permetés demostrar que l'algorisme és correcte. **(1 punt)**

1.3 Expliqueu en quins casos es pot demostrar la postcondició de l'acció *fusio_ord* a partir de l'invariant i de la condició de terminació d'aquest bucle. **(0.5 punts)**

1.4 Completeu la implementació de l'acció **ordena**, que ordena una llista d'enters utilitzant el mètode de *ordenació per fusió* (*mergesort*). Per exemple, si `t={4,-2,6,5,1}` abans de la crida `ordena(t)`, després d'aquesta el seu valor serà `t={-2,1,4,5,6}`. **(1.5 punts)**

void *ordena*(*list* <**int**>& *t*) {
/* Pre: t = T */
/* Post: t conté una permutació dels elements de T. t està ordenada en ordre creixent. */

Observació: Per resoldre alguns apartats es poden utilitzar les següents versions de l'operació `splice` de la classe `list`, que permeten transferir elements de la llista x al paràmetre implícit, cosa que altera la mida d'ambdues llistes.

void *splice* (*iterator p*, *list* & *x*);
/* Pre: p apunta a un element del paràmetre implícit o a this→end(). */
/* Post: Transfereix al paràmetre implícit tots els elements de la llista x i els insereix davant de la posició a la qual apunta p. x.size() = 0. */

void *splice* (*iterator p*, *list* & *x*, *iterator it*);
/* Pre: p apunta a un element del paràmetre implícit o a this→end(). it apunta a un element de x. */
/* Post: Transfereix al paràmetre implícit l'element de la llista x al que apunta it i l'insereix davant de la posició a la qual apunta p. x.size() = X.size() − 1. */

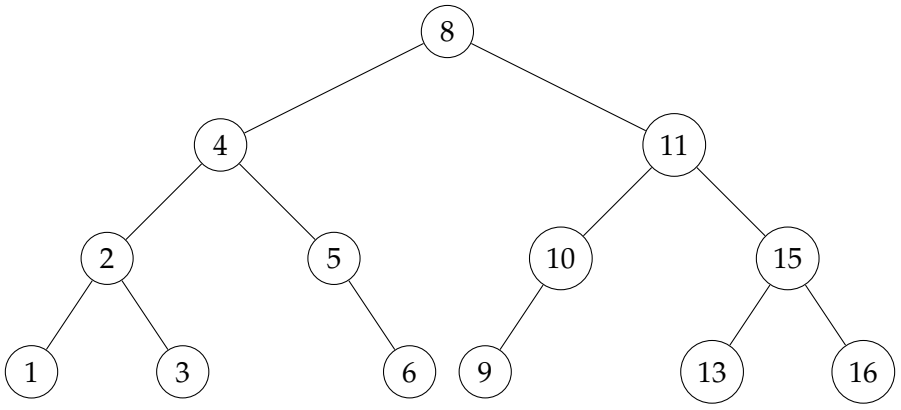
void *splice* (*iterator p*, *list* & *x*, *iterator first* , *iterator last*);
/* Pre: p apunta a un element del paràmetre implícit o a this→end(). first apunta a un element de x. last apunta a un element de x o a x.end(). */
/* Post: Transfereix al paràmetre implícit els elements de la subllista de x que comença en l'element al qual apunta first i acaba en l'element anterior al qual apunta last i els insereix davant de la posició a la qual apunta p. x.size() = X.size() − mida subllista transferida. */

Aquestes operacions no involucren la construcció o la destrucció de cap element. Els iteradors que apuntaven als elements transferits segueixen apuntant a aquests elements, però després de l'aplicació de l'operació `splice` aquests iteradors iteren en el paràmetre implícit, i no en la llista x, és a dir, es mouen (fent servir els operadors ++ i --) pel paràmetre implícit.

Cognoms	Nom	DNI

Problema 2 (5 punts)

Donat un arbre binari T i un enter $k \geq 0$, denotem mitjançant $N_k(T)$ el nombre de nodes de T de nivell k (equivalentment, de profunditat k). Per exemple, si T és l'arbre de la figura



llavors $N_0(T) = 1$, $N_1(T) = 2$, $N_2(T) = 4$ i $N_3(T) = 6$. Observeu que $0 \leq N_k(T) \leq 2^k$ per a tot nivell $k \geq 0$ de qualsevol arbre binari T (en particular, l'arrel és al nivell 0), i que $N_k(T) = 0$ per a tot nivell $k \geq$ alçada(T). Recordeu que la *alçada* d'un arbre binari T es defineix de la manera següent

- Si T és buit, alçada$(T) = 0$.
- En un altre cas, alçada$(T) = 1 +$ màxim(alçada$(T.left())$, alçada$(T.right())$).

Responeu als següents apartats tenint en compte les definicions anteriors.

1. **Implementeu una funció recursiva** `nivell`(T, k) que, donats un arbre binari T i un enter $k \geq 0$, retorni $N_k(T)$. **(1 punt)**

2. **Feu un disseny recursiu de la funció** *profile*, especificada a continuació. En aquest apartat **es recomana fer servir una funció d'immersió** que eviti la creació i destrucció de vectors en cada crida recursiva. Si feu servir una funció d'immersió, heu d'especificar la seva precondició i postcondició.

   ```
   // Pre: h = alçada(T)
   vector<int> profile(const BinTree<int>& T, int h);
   // Post: v.size() = h, i v[k] = N_k(T) per a tot 0 ≤ k < h.
   ```

   Per exemple, el resultat de la crida `profile(T,4)` amb l'arbre T de la figura ha de ser el vector `[1,2,4,6]`. **Observació:** La vostra solució **ha de visitar cada node de l'arbre T un sol cop com a màxim**. En particular, no ha de fer crides a la funció `nivell` de l'apartat anterior. **(2.5 punts)**

3. Donat un arbre binari T el *nivell de saturació* de T és el nombre de nivells j per als quals $N_j(T) = 2^j$. Per exemple, el nivell de saturació de l'arbre T de la figura és 3.

   Feu un disseny recursiu de la funció *nsat* tenint en compte les restriccions següents. **(1.5 punts)**

   ```
   // Pre: Cert
   int nsat(const BinTree<int>& T);
   // Post: El resultat és el nivell de saturació de T.
   ```

   - La vostra solució ha de visitar cada node de l'arbre T un sol cop com a màxim.
   - Per a aquest apartat **no es pot utilitzar cap funció d'immersió**, és a dir, en el cos de la funció *nsat* no hi pot haver crides a una altra funció auxiliar.