Problema 1:

```
void arregla(list<int>& l) {
    list<int>::iterator it1, it2;
    int max;
    it1 = l.begin();
    max = *it1;
    ++it1;
    it2 = it1;
    while (it2 != l.end()) {
       if (*it2 > max) {
          max = *it2;
          if (it1 != it2) {
             l.insert(it1,*it2);
             it2 = l.erase(it2);
          }
          else {
             ++it1;
             ++it2;
          }
       } else {
          ++it2;
       }
    }
}
```

Justificació de que el bucle manté les parts 3, 4 i 5
de l'invariant.

Observem primer que en qualsevol cas l'iterador it2 en
acabar la iteració apunta a l'element següent al que apuntava
al principi de la iteració.

- Si *it2 > max, per 5, *it2 és el màxim trobat fins el moment.
Com que it2 avança una posició, fent *it2 = max mantenim 5.

- Si a més it1 != it2, l'insert i el delete afegeixen *it2 com a
darrer màxim local just abans d'it1 (per mantenir 3), i asseguren que
*it2 (que és màxim local) no apareix entre it1 i it2 (per mantenir 4).

- Si *it2 > max però it1 == it2, és que fins ara o s'havia
trobat cap element que no fos màxim local. Avançant tant it1 com it2
automàticament queda posat *it2 com a darrer màxim local abans que
it1 (mantenim 3) i mantenim 4 perquè tampoc *it2 és màxim local.

- si pel contrari *it2 >= max, *it2 no és màxim local.
En aquest cas, 3 es manté automàticament perquè els màxims locals
trobats segueixen sent els trobats fins a la iteració anterior
(mantenim 3), i avançant it2 assegurem que *it2 queda entre it1 i it2
com a darrer no-màxim-local, per tant mantenim 4.

Problema 2:

```
 int i_nombre_ponderats(Arbre<int>& a, int sum_asc, int& sum_desc)
 /* Pre: a = A */
 /* Post: el resultat es el nombre d'elements d'A que son mes grans que
    la suma de sum_asc amb la suma dels seus ascendents i mes petits que
    la suma dels seus descendents; sum_desc es la suma dels elements d'A
*/
{
    if (a.es_buit()) {
       sum_desc = 0;
       return 0;
    } else {
       int arr = a.arrel();
       Arbre <int> fe, fd;
       int se, sd, ne, nd;
       a.fills(fe,fd);
       ne = i_nombre_ponderats(fe, sum_asc + arr, se);
       nd = i_nombre_ponderats(fd, sum_asc + arr, sd);
       sum_desc = se + sd + arr;
       if (sum_asc < arr and arr < sum_desc - arr) {
           return ne + nd + 1;
       } else {
           return ne + nd;
       }
    }
 }

int nombre_ponderats(Arbre<int>& a) {
  int sum_desc;
  return  i_nombre_ponderats(a, 0, sum_desc);
}
```