PROBLEMA 1

1.1

Siguin n1 i n2 el nombre d'elements dominadors d'A1 i A2.
Si x és més gran que tots els elements d'A1 i d'A2,
el nombre d'elements dominadors d'A és n1+n2+1, i altrament
és n1+n2.

Es pot veure que x és més gran que tots
els elements d'A1 i A2 si és més gran que els elements
màxim d'A1 i d'A2. Si algun dels subarbres A1 i A2 és buit
el seu element màxim no està definit i no entra en la comparació.
En particular, si A és una fulla (A1 i A2 són buits) llavors
hi ha un únic element dominador, l'arrel d'A.

1.2

Per eficiència, intentarem no haver de calcular separadament
els màxims dels fills amb una altra funció (que recorreria un altre cop l'arbre)
sinó aprofitar la mateixa crida recursiva que calcula n1 i n2
per retornar el valor màxim d'A a qui l'hagi cridat. Cal tenir
en compte que el màxim valor d'un arbre buit no està definit.

```
int i_num_dominadors(Arbre<int> &a, int& max_tot);
/* Pre: a = A i tot node d'A té o 0 o 2 fills */
/* Post: el resultat retornat indica el nombre de nodes dominadors d'A
   i, si A no és buit, max_tot és el valor màxim dels nodes d'A */
```

1.3

```
int num_dominadors(Arbre<int> &a) {
    int max_tot;
    return i_num_dominadors(a, max_tot);
}
```

1.4

```
int i_num_dominadors(Arbre<int> &a, int& max_tot) {
   if (a.es_buit()) {
       // no cal que max_tot valgui res definit però l'arbre buit té 0 dominadors
       return 0;
   } else {
       Arbre<int> a1, a2;
       int x = a.arrel();
       a.fills(a1,a2);
       if (a1.es_buit()) {
          // a és una fulla: l'arrel és l'únic element i és dominador
          max_tot = x;
          return 1;
       } else {
          int max1, max2;
          // a és un node amb dos fills no buits
          int n = i_num_dominadors(a1, max1);
          n += i_num_dominadors(a2, max2);
          if (x > max1 and x > max2) ++n;
          max_tot = max(x, max(max1, max2));
          return n;
       }
   }
}
```

Hem aprofitat la funció max de STL (només cal fer #include <algorithm>).

PROBLEMA 1

2.1

Es donen tres solucions tipus amb la part novedosa dels seus
respectius invariants.

A)
```
  int k = 1;
  int ultima_variacio = v[1] - v[0];
  while (k < v.size()-1) {
  // Inv: ... ultima_variacio conté l'última diferència no zero entre dos elements consecutius de v[0..k]
    if (ultima_variacio > 0 and v[k] > v[k + 1])
      maxims.push(make_pair(k, v[k]));
    if (v[k] != v[k + 1])
      ultima_variacio = v[k+1] - v[k];
    ++k;
  }
```

B)
```
  int k = 1;
  int j = 1;
  while (k < v.size()-1) {
  // Inv: ... 1<=j<=k, v[j]!=v[j-1], tots els elements de v[j..k] són iguals
    if (v[j] > v[j-1] and v[k] > v[k + 1])
      maxims.push(make_pair(k, v[k]));
    if (v[k] != v[k+1])
      j = k+1;
    ++k;
  }
```

C)
```
  int k = 1;
  bool creix = v[1] > v[0];
  while (k < v.size() - 1) {
  // Inv: ... creix indica si l'última diferència no zero entre dos elements consecutius de v[0..k] és creixent
    if (creix and v[k] > v[k + 1])
      maxims.push(make_pair(k, v[k]));
    if (v[k] != v[k + 1])
      creix = v[k + 1] > v[k];
    ++k;
  }
```

D) Amb un bucle intern que avança fins a trobar el primer element diferent de v[k]. Caldrà a 2.2 justificar també el bucle intern.

2.2 Justifiquem només la variant A)

Invariant complet:

1 <= k < v.size(),
maxims conté els parells (posició,valor) dels màxims locals de v[0..k-1] ordenats creixentment per posició,
ultima_variacio conté l'última diferència no zero entre dos elements consecutius de v[0..k]

Justificacions:

a) les inicialitzacions estableixen l'invariant:

k=1, juntament amb v.size()>=2 en la Pre, compleix la restricció de rang 1 <= k < v.size();
maxims és una cua buida (per la Pre) i per definició no hi cap màxim local a v[0..0];
ultima_variacio=v[1]-v[0] es pot calcular (els indexos són vàlids atès que v.size()>=2),
és diferent de zero (per la Pre) i és l'última diferència no zero entre dos elements consecutius de v[0..1]

b) El cos del bucle manté l'invariant:

la condició del bucle i l'invariant impliquen que es continua complint la restricció de rang de k després d'incrementar-la al final del bucle;
abans de poder incrementar la k, cal que maxims contingui els parells dels màxims locals de v[0..k], i com que per l'invariant ja els contenen pel subvector v[0..k-1], cal afegir (k,v[k]) a la cua maxims si v[k] és un màxim local;
això es el que fa la primera instrucció if del bucle per la definició de màxim local i per la definició de la variable ultima_variacio a l'invariant;
finalment, cal actualitzar ultima_variacio perquè contingui l'última diferència no zero entre dos elements consecutius de v[0..k+1] (abans d'incrementar la k) i això és el que fa el segon if del bucle.

tot els accessos a v són vàlids perquè les posicions k i k + 1, gràcies a l'invariant del bucle i la condició del bucle,
són valors entre 0 i v.size() - 1.

c) En acabar el bucle, per l'invariant, se satisfà la Post:

la restricció de rang sobre k en l'invariant juntament amb la negació de la condició del bucle implica que k = v.size()-1 al sortir del bucle; llavors, donat que per definició v[v.size()-1] no pot ser màxim local, i que per l'invariant, maxims conté a l'acabar el bucle els parells corresponents als màxims locals (ordenats creixentment per posició) de v[0..v.size()-2], vol dir que maxims conté respectivament els parells corresponents als màxims locals de v (ordenats creixentment per posició).