Write a C++ function, which returns true IFF the supplied array's contents represent a legal winner tree:

Theory: by definition, a winner tree must be **complete** and (as shown in the diagram) and a node must contain the same value as one of its children.

bool is\_winner\_tree(int\* tree, unsigned int n, int unused){

if(n<2) return true;

//check for completeness

//if unused slot left of a used slot, the tree is not complete

for (int i=1; i!=n; ++i){

if((tree[i-1]==unused) && (tree[i]!=unused)){

return false;

}

//check parents for same value as one of its children

for(int i =0; i!=n; ++i){

int left = 2\*i;

int right = left+1;

if(left<n){ //there is a slot for a left child

if(right < n){ //there is a slot for a right child

if( //parent matches a child

((tree[i]==tree[left]) || (tree[i]==tree[right]))

&& //the children are different (or both unused slots)

(tree[left] == unused || (tree[left]!=tree[right])))

continue;

return false;

}else{

if(tree[i]==tree[left]) //parent matches only child

break; //we just checked the last child

return false;

}

}

return true;

}

Write a C++ function, which returns true IFF the supplied array's contents represent a legal min heap-ordered binary tree.

Theory: by definition, the children of each node in a heap-ordered tree may not have a higher priority than the node. The tree need not be complete.

We'll traverse the "tree" from the root; we'll need to ensure that the "tree" isn't really a forest (a used slot with an unused slot parent), and that children don't have higher priorities.

bool is\_heap\_ordered(int\* tree, unsigned int n, int unused){

if(n==0) return true; //this check need not appear in a student's solution

//ASSUMING check for min heap-ordered binary tree

//check parents for same value as one of its children

for(int i =0; i!=n; ++i){

int left = 2\*i;

int right = left+1;

if(left < n){ //there is a slot for a left child

if((tree[i]==unused) && (tree[left]!=unused))

return false; //illegal: forest, not a tree

if((tree[i]!=unused) && (tree[i]>tree[left]))

return false; //not min-heap ordered

}

if(right < n){ //there is a slot for a right child

if((tree[i]==unused) && (tree[right]!=unused))

return false; //illegal: forest, not a tree

if((tree[i]!=unused) && (tree[i]>tree[right]))

return false; //not min-heap ordered

}

}

return true;

}