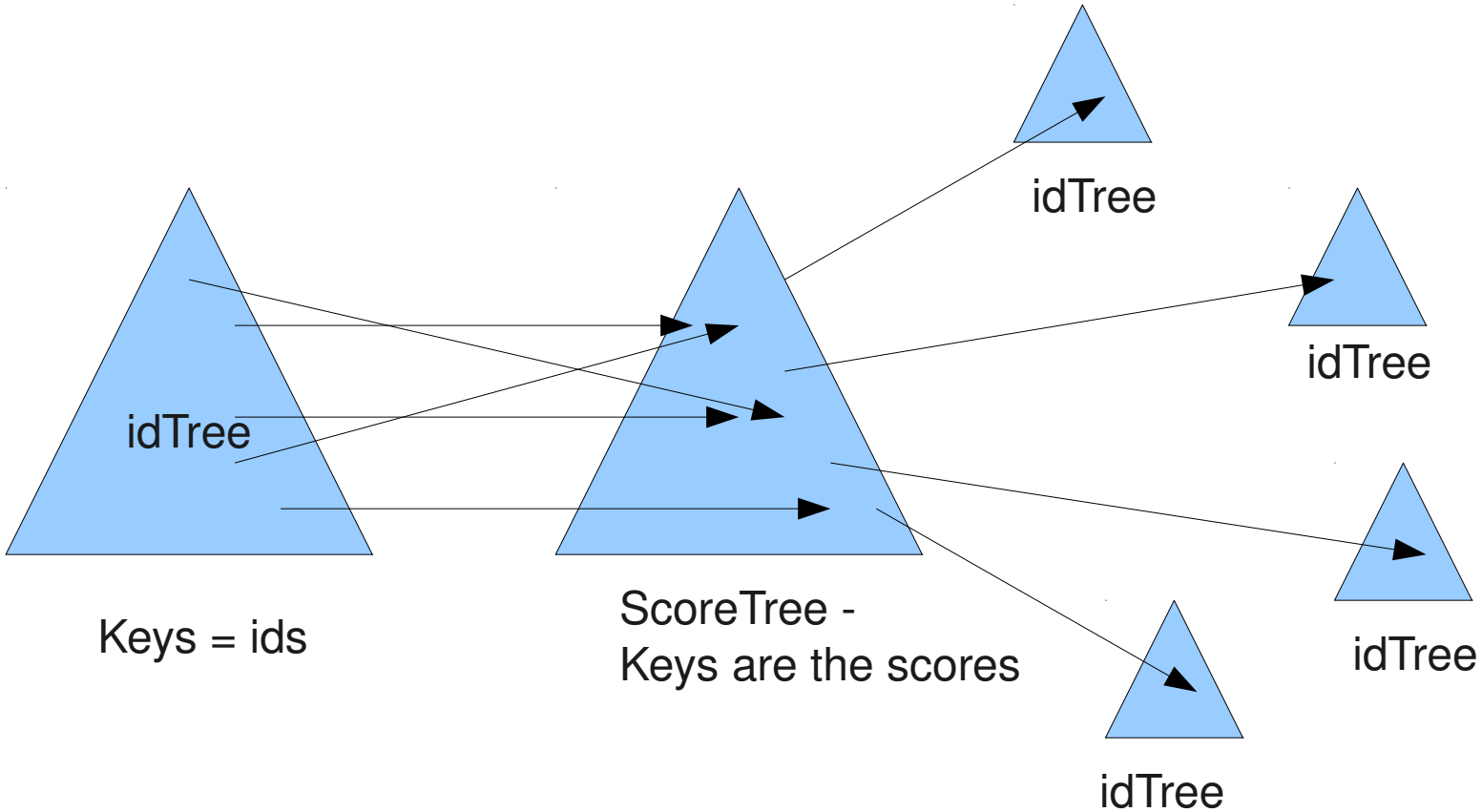
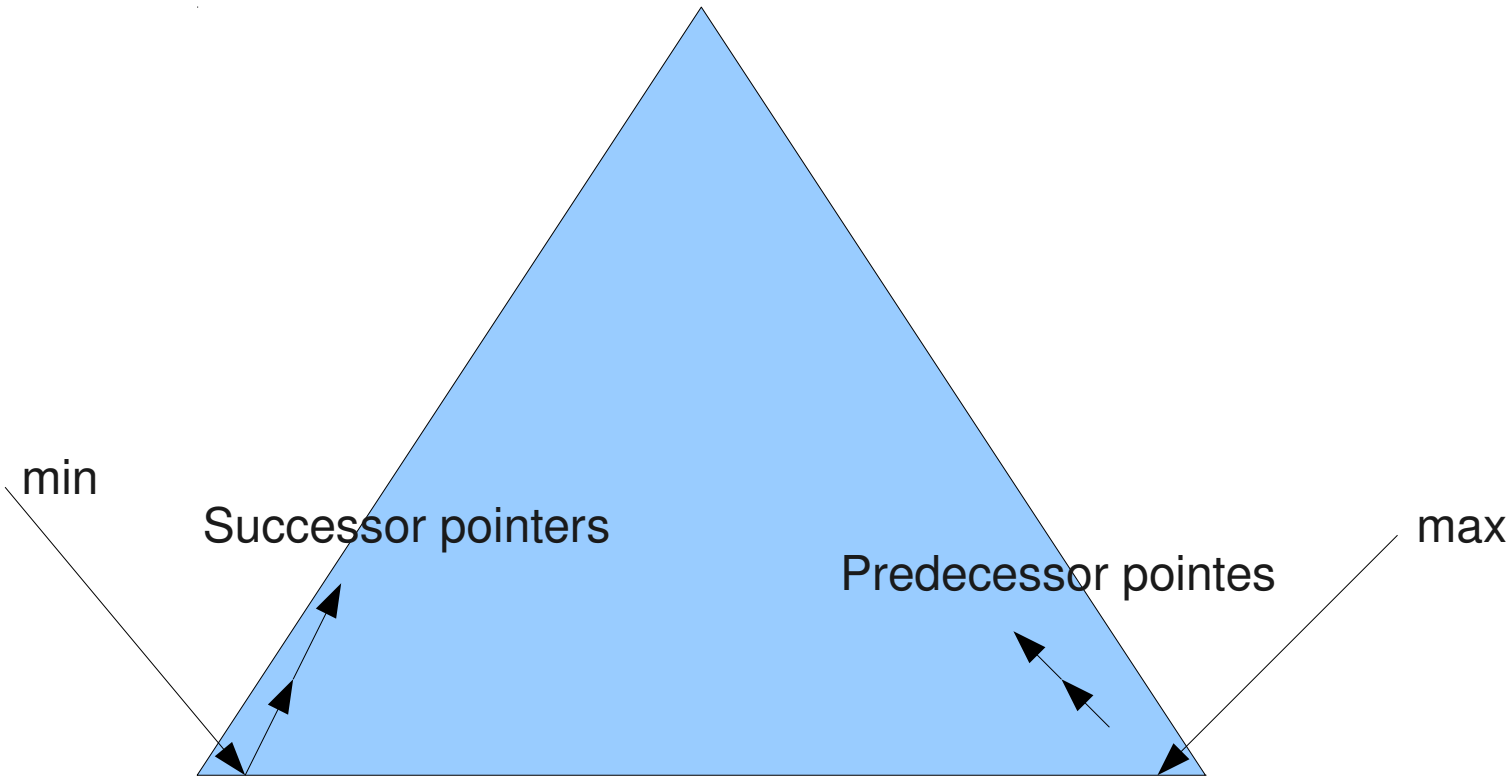


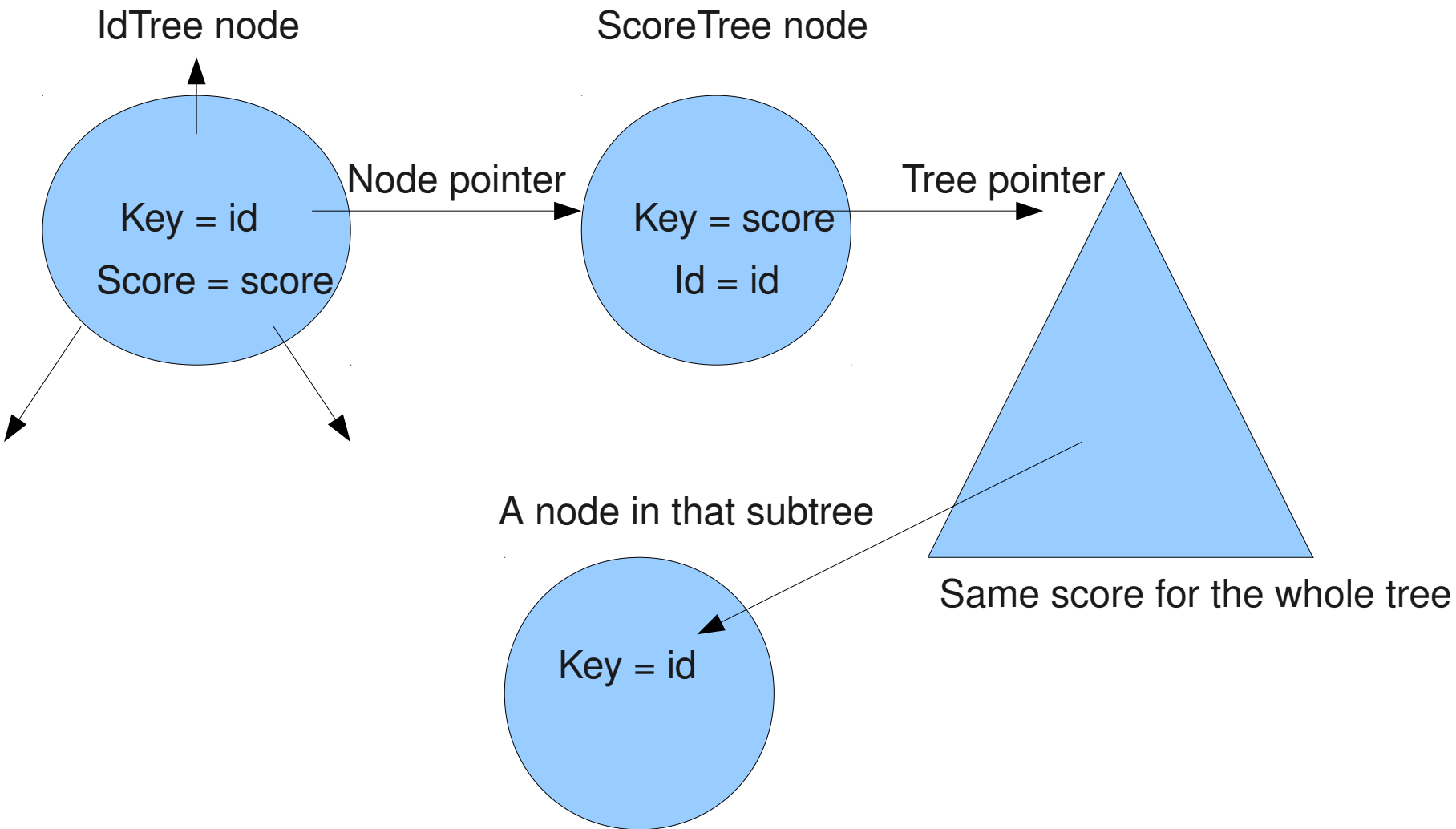
All tree instances are of class AVLTree
Idtree hosts nodes that are sorted by the players ids
They have pointers to the corelating score in the scoretree
Each score node points to an id tree with cloned nodes of the ids
That share the same score and sorts them according to the ids
All nodes In those tree instances are of class Node , there is a var
Redundancy but its a small price to pay for reusability and abstraction
Without usage of inheritance and polimorphism (which gave the advantage of
A smaller class package) , **(this datastructure also answers part B)**

AVLTree class focus



Also has a size field it keeps in check

Node instance example focus



Complexity calculations

Insert-player(id) - insert to idtree O(lgn) make scoreNode and draw a pointer to it O(1)
Insert to scoreTree O(lgn) and insert to subTree O(lgn) . Total complexity
O(lgn)
Get-player(id) – search idtree O(lgn) follow pointer to scoreTree O(lgn) get data from nodes
O(1) , total complexity O(lgn).
Get-rank(id) – search idtree O(lgn) follow pointer if the subtree is <= 1 alias scorenode
Else search in subtree O(lgn) and alias node O(1)
Get max from Score tree O(1) walk back from max to aliased scorenode and count steps
(in subtrees aswell the same way only from min forward) , total complexity O(steps/rank)
Get-top(y) – get max pointer from scoreTree O(1) walk back “y” steps(subtrees will be from min
Farward) and as you walk
Collect the nodes into a row data struct O(y) then spit out the info in those nodes in order
O(y), total complexity O(y)
Insert-Game(id1,tactic1,id2,tactic2) – search idTree for ids O(lgn) alias them O(1)
Remove from idtree scoretree and subtree O(lgn) , check who won O(1), extract old data
From aliased nodes O(1),Insert with new data into idtree O(lgn) and scoretree O(lgn),
total complexity O(lgn). (which is also the answer to part B of the homwork assingment)