•1) Write a Java class that implements a generic Heapsort of an array of Objects (of any type). Your method must receive two parameters: the array and an extra object containing the method to compare elements of the array. You must test your method with objects of type String. You will also use this method for the rest of the homework.

•2) This problem deals with the Tree Isomorphism Problem. You will receive two trees from a file and you must determine whether they are isomorphic, meaning that they are the same tree upto relabelling of the vertices. Briefly this is what your program should do:

A) read two trees from a file
   format: root(buy(c)(d))(e(fit))(go)
   resulting tree:

   root
   /  
  buy  e  go
  /  
 c  d  fit

B) Re-root each tree at its center.
C) Assign an integer label to each vertex
D) The trees are isomorphic iff their sets of labels are equal.
E) If the trees are isomorphic then print out a 1-1 mapping between the vertices.

Algorithm Isomorphic(T₁, T₂)

Input: Rooted trees T₁ and T₂.
Output: If T₁ is isomorphic to T₂ then a 1-1 mapping between their vertices proving that they are isomorphic; otherwise, an empty mapping.

C₁ ← FindCenter(T₁)
ReRoot(T₁, first vertex in C₁)
C₂ ← FindCenter(T₂)
ReRoot(T₂, first vertex in C₂)

M ← RootedIsomorphic(T₁, T₂)
IF M is empty and size(C₂) > 1 THEN
   ReRoot(T₂, second vertex in C₂)
   RootedIsomorphic(T₁, T₂)

return M
**Algorithm** FindCenter(T)

**Input:** Rooted tree T.
**Output:** Returns the set of vertices at the center of T; that is, the set of vertices furthest from a leaf. The set has size either one or two.

CalculateDistancesFromLeaves(T)
max ← max{ distance(v) | v in T}
return { v | v in T and distance(v) = max }

**Algorithm** CalculateDistancesFromLeaves(T)

**Input:** Rooted tree T.
**Output:** For each vertex v in T, calculates v.distance which is the length of the shortest path between w and a leaf in T.

Use a postorder traversal of T to initially set
v.distance ← min{ w.distance + 1 | w is a child of v }

Use a preorder traversal of T to finally set
v.distance ← min{ v.distance, v.parent().distance + 1 }

**Algorithm** ReRoot(T, v)

**Input:** T is a rooted tree and v a vertex in T.
**Output:** T rooted at v.

T.setRoot(v)
p ← T.parent(v)
WHILE p ≠ null DO
remove v from p's list of children
add p to v's list of children
g ← T.parent(p)
make p's parent v
v ← p
p ← g
Algorithm RootedIsomorphic($T_1$, $T_2$)

Input: $T_1$ and $T_2$ are rooted trees.
Output: If $T_1$ is rooted-isomorphic to $T_2$ then produces a 1-1 mapping between their vertices, proving that they are rooted-isomorphic; otherwise, an empty mapping.

$$h \leftarrow T_1.\text{height}()$$
IF $h \neq T_2.\text{height}()$ THEN
    return empty mapping
L_1 \leftarrow \text{an array of } h+1 \text{ empty sequences}
L_2 \leftarrow \text{an array of } h+1 \text{ empty sequences}
L_1[h] \leftarrow \text{Initialize($T_1$, } h)\)
L_2[h] \leftarrow \text{Initialize($T_2$, } h)\)
FOR $i \leftarrow h-1, h-2, ..., 0$ DO
    L_1[i] \leftarrow \text{LabelLevel($T_1$, } L_1[i+1], i)\)
    L_2[i] \leftarrow \text{LabelLevel($T_2$, } L_2[i+1], i)\)
    IF $L_1[i].\text{size}() \neq L_2[i].\text{size}()$ THEN
        return empty mapping
    FOR $i \leftarrow 0, 1, ..., L_1[i].\text{size}()$ DO
        v \leftarrow \text{vertex } i \text{ in } L_1[i]\)
        w \leftarrow \text{vertex } i \text{ in } L_2[i]\)
        IF orderedlabels(v) \neq \text{orderedlabels}(w) THEN
            return empty mapping
    M \leftarrow \text{empty sequence}
GenerateMapping($T_1.\text{root}(), T_2.\text{root}(), M)\)
return M

Algorithm Initialize($T$, $h$)

Input: Rooted tree $t$ of height $h$.
Output: Initializes the label, orderedchildren sequence and orderedlabels sequence of each vertex in $T$. Returns $L$, a sequence containing all vertices at depth $h$ in $T$.

$$L \leftarrow \text{an empty sequence}$$
FOR each vertex $v$ in $T$ DO
    v.label \leftarrow 0
    v.orderedchildren \leftarrow \text{empty sequence}
    v.orderedlabels \leftarrow \text{empty sequence}
    IF $T.\text{depth}(v) = h$ THEN
        L.insert(v)
return L
Algorithm LabelLevel(T, P, d)

Input: Rooted tree T, integer d between 0 and T.height(), and sequence P containing all vertices at depth d+1 in T arranged in ascending order of their integer labels.
Output: Calculates integer label for each vertex v at depth d in T so that if v and w are vertices in T at depth d then v.label = w.label if and only if the subtrees rooted at v and w are isomorphic.
Returns sequence L containing all vertices at depth d in T arranged in ascending order of their integer labels.

L ← empty sequence
FOR i ← 0, 1, ..., P.size()-1 DO
    v ← vertex i in P
    IF T.parent(v).orderedlabels.isEmpty() THEN
        L.insert(T.parent(v))
        T.parent(v).orderedlabels.insertAtBack(v.label)
        T.parent(v).orderedchildren.insertAtBack(v)
    Sort L in descending order of the orderedlabels of each vertex
    FOR each vertex v in L DO
        v.label ← k where v.orderedlabels is the k\textsuperscript{th} largest in L

Algorithm GenerateMapping(v, w, M)

Input: Vertices v and w, roots of isomorphic subtrees, and sequence M.
Output: Adds 1-1 mapping between the vertices in the subtree rooted at v and the vertices in the subtree rooted at w proving that the subtrees are isomorphic.

M.insert(pair v,w)
FOR i ← 0, 1, ..., v.orderedchildren.size()-1 DO
    x ← vertex i in v.orderedchildren
    y ← vertex i in w.orderedchildren
    GenerateMapping(x, y, M)