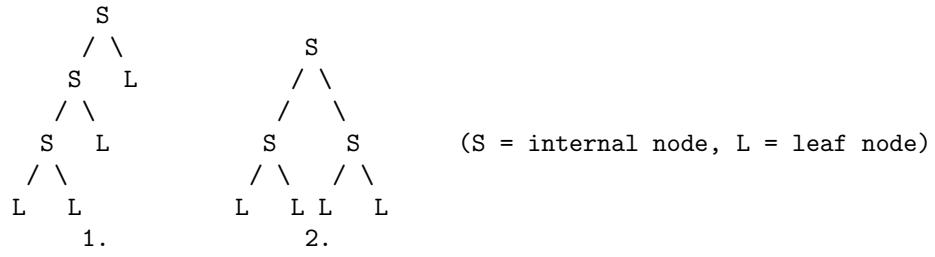
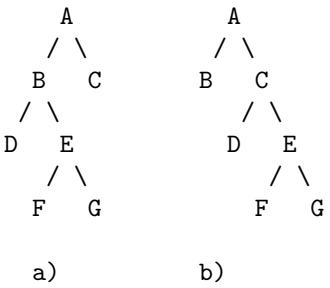
Data Structures 2018
Exercise 6, solutions (Week 42)

1.-2. The solution can be found in a Java file from the course home page.

3.


The worst case is presented in Figure 1 above. Thus the maximum height of a binary tree is $(n-1)/2$. The best case is given in Figure 2. Now the minimum height is $\log_2(n+1)-1$.

4.


5. Algorithms 1–7. It is assumed that we are dealing with an array representation of a proper binary tree (S), in which missing nodes have been replaced with null references (n). In addition, it is assumed that the zeroth index contains a null reference, whereupon the first index contains the root node. From the viewpoint of the assignment, the most important thing was to consider the various methods as operations of a binary tree. Therefore, the following pseudocodes do not consider the validity of the parameters (e.q. is the binary tree S empty, is the specified index out of bounds, are the computed indices out of bounds, and so forth.).

Algorithm 1 Returns the value of the root node.

```
root(S)
  return S[1]
```

Algorithm 2 Returns the value of the kth node's parent node.

```
parent(S,k)
  return S[⌊(k/2)⌋]
```

Algorithm 3 Returns the value of the kth node's left child.

```
leftChild(S,k)
  return S[2k]
```

Algorithm 4 Returns the value of the kth node's right child.

rightChild(S,k)

return $S[2k+1]$

Algorithm 5 Returns logical true if the specified node is internal.

isInternal(S,k)

if $S[k]$!= n **and** (leftChild(S,k) != n **or** rightChild(S,k) != n) **then**
return true
else
return false
end if

Algorithm 6 Returns logical true if the specified node is external.

isExternal(S,k)

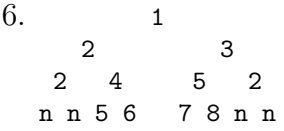
if $S[k]$!= n **and** leftChild(S,k) = n **and** rightChild(S,k) = n **then**
return true
else
return false
end if

Algorithm 7 Returns logical true if the specified node is the root node.

isRoot(S,k)

if $S[k]$!= n **and** parent(S,k) = n **then**
return true
else
return false
end if

6.



7.-8. The table and the number of elements (currentELements) in it have to be stored in both a) and b).

Algorithm 8 removeMinElement ()

a)
```
if currentElements > 0 then
  result = table[0]
  for i = 1 to currentElements-1 do
    table[i-1] ←table[i]
  end for
  currentElements = currentElements-1
else
  error "The queue is full"
end if
return result
```

Algorithm removeMinElement removes an element from the beginning of the array and shifts rest of the elements.

Algorithm 9 insertItem (x)

```
if currentElements < maxElements then
  i ← currentElements
  while i > 0 and table[i − 1] > x do
    table[i] ←table[i − 1]
    i ← i − 1
  end while
  table[i] ← x
  currentElements ← currentElements +1
else
  error "The queue is full"
end if
```

Algorithm insertItem looks for the position, makes some room for the new key, shifts rest of the keys forward, and finally inserts the new key to its place.

Algorithm removeMinElement looks for the smallest element in the array, makes a temporary copy of it, replaces the original element with the last element, decreases the size of the array by one, and finally returns the temporary element.
Algorithm inserItem inserts a new element to the end of the array. This increases the size of the array by one.

Algorithm 10 removeMinElement ()

b)
```
if currentElements > 0 then
  minIndex ←0
  min ←table[0]
  for i ←1 to currentElements-1 do
    if table[i] < min then
      minIndex = i
      min = table[i]
    end if
  end for
  table[minIndex] ←table[currentElements-1]
  currentElements ←currentElements-1
else
  error "The queue is full"
end if
return  min
```

Algorithm 11 insertItem (x)

```
if currentElements < maxElements then
  table[currentElements] = x;
  currentElements = currentElements + 1
else
  error "The queue is full"
end if
```