```
TDA416 2014-03-11
Problem 1
a) nej det uppfyller inte vänsterbalansen (fyllt utom på nedersta nivån från höger)
och inte är det partiellt ordnat heller, se tex roten och dess barn.
b) nej 9 är större än 8
c) ja abs(hl-hr)<=1 i alla noder
d) 6, 4, 2, 1, 3, 5, 8, 7, 9, 10, 11 (rot, vä delträd, hö delträd)
e) 6, 4, 8, 2, 5, 7, 10, 1, 3, 9, 11 (nivå för nivå)
f) 2^h-1 fullt träd, varje nivå dubblar antalet noder utom första nivån
g) h om trädet är en lista
h) 1 se ovan
i) O(str.length)  eg. O(str.length x "hello".length)
och skall man vara ännu noggrannare så är det (n-m+1)*m
där n = str.length och m = hello.length
j) Syntaxfel som jag inte såg: en parentes fattas
Men komplexiteten bör bli densamma om man inte ändrar mycket
En möjlighet:
boolean checkString(String[] strs, String str) {
    for (int i=0; i<strs.length; i++) {
        if (strs[i]==str) {
            return true;
        } else {
            return false;
        } // tillagd
    }
}
En annan möjlighet
boolean checkString(String[] strs, String str) {
    for (int i=0; i<strs.length; i++) {
        if (strs[i]==str) {
            return true;
        } else  // bortagen
            return false;
    }
}
komplexitet för bägge: ett varv i loopen med konstant tid
"==" är pekarjämförelse som tar konstant tid.

Man får ändra rätt mycket för att få något annat tex:
boolean checkString(String[] strs, String str) {
    for (int i=0; i<strs.length; i++) {
        if (strs[i]==str) {
            return true;
        }
    }
    return false;
}
Komplexitet: O(strs.length)
============================================================
Problem 2        jun
           /             \
        dec                nov
      /    \             /     \
   aug      jan       maj      okt
 /    \    /    \    /    \    /    \
apr  xxx  feb  jul  maj  xxx  xxx  sep

Trädet är inte unikt. Längsta vägen är 3.
1) sortera: apr, aug, dec, feb, jan, jul, jun, maj, mar, nov, okt, sept
2) D&C: börja i mitten av den sorterade sekvensen (jul/jun spelar inte så stor roll)
3) applicera 2) rekursivt på de bägge halvorna
Sorteringen tar O(nlogn) + D&C O(n)  [ T(n) = 2T(n/2)+c ]
============================================================
Problem 3
se bok/OH bilder för algoritmer
a) A, F, B, C, D, E, M, L, J, K, H, G, I
b) dfs(M) = M, L, J, G, I, K, H, dfs(A) = A, B, C, D, E, dfs (F) = F
============================================================

```

```
Problem 4
private Node<Activity> head = null;
private int size = 0;

public void add(Activity fresh) {
    if (fresh == null) {
        throw new IllegalArgumentException("null activity");
    }
    head = addR(fresh, head);
} // **********************************
private Node<Activity> addR(Activity fresh, Node<Activity> head) {
    if (head == null ||
        fresh.end <= head.data.start) { // insert "first"
        size++;
        // clone fresh before add to list
        Activity clonedFresh = new Activity(fresh.data, fresh.start, fresh.end);
        return new Node<Activity>(clonedFresh, head);
    } else if ( fresh.start >= head.data.end ) { // keep searching
        head.next = addR(fresh, head.next);
        return head;
    } else {
        return head; // can't do it
    }
} // end addR
// **********************************
public Activity remove() {
    // always first element,
    // no need to clone here since we are removing from list
    // what to do if if head==null? default is to use NullPointerException
    Activity item = head.data;
    head = head.next;
    size--;
    return item;
} // **********************************
public int getSize() {
    return size;
}

Insättningssortering är det troligast att man använder på en sorterad länkad lista.

```

```
============================================================
Problem 5
För att kunna avgöra om listan ändras under iterationen med iteratorn så lägger man
… in en variabel int modCount = 0; Sedan ökar man denna variabel så fort en ändring av
… listan sker. (modCount++;)
I iteratorn har man en variabel expectedModCount som lagrar värdet av modCount
när iteratorn skapas. Sedan kan man jämföra modCount och expectedModCount innan
iteratorn returnerar ett objekt. Är dom olika så har en otillåten förändring skett.

Sen ändras class huvudet till
public class ActivityList implements Iterable {...
också skall man ha en iterator metod såklart.

private class ActivityListIterator implements Iterator<Activity> {
    private Node<Activity> currentNode = head;
    private Node<Activity> lastReturned = null;
    int expectedModCount = -1;

    public ActivityListIterator() {
        currentNode = head;
        lastReturned = null;
        expectedModCount = modCount;
    } //*******************************
    public boolean hasNext( ) {
        checkForComodification();
        return (currentNode != null) ;
    } //*******************************
    public Activity next( ) {
        if ( !hasNext() ) {
            throw new NoSuchElementException();
            // or return null;
        } else {
            lastReturned = currentNode;
            currentNode = currentNode.next;
            return lastReturned.data;
        }
    } // ****************************
    public void remove( ) {
        throw new UnsupportedOperationException();
    } // ****************************
    final void checkForComodification() {
        if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
    } //*******************************
} // end class ActivityListIterator
============================================================

```