Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2012-05-25**
Program	**DAI2+I2**
Läsår	**2011/2012, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (4+6 p)

a)

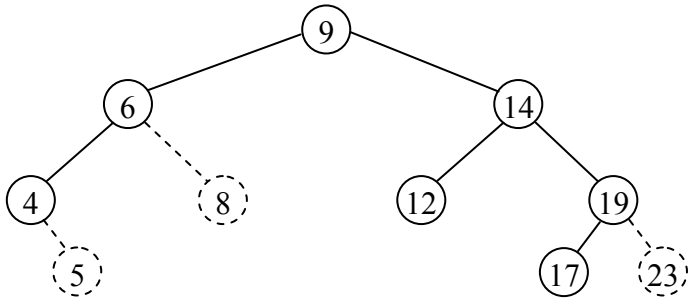
```
public static boolean isFull(TreeNode t) {
    if ( t == null )
        return true;
    else
        return TreeNode.height(t.left) == TreeNode.height(t.right) &&
               isFull(t.left) && isFull(t.right);
}
```

b)

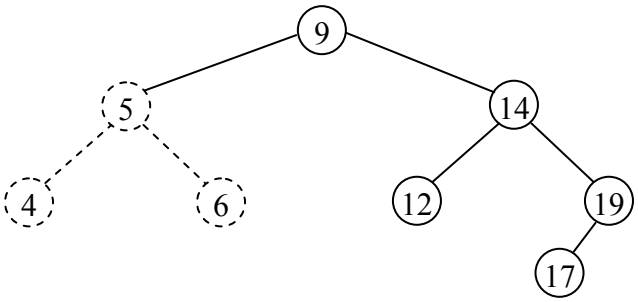
```
public static boolean isComplete(TreeNode t) {
    if ( t == null )
        return true;
    else {
        final int hl = TreeNode.height(t.left),
                  hr = TreeNode.height(t.right);
        return
            hl == hr && isFull(t.left) && isComplete(t.right) ||
            hl == hr + 1 && isComplete(t.left) && isFull(t.right);
    }
}
```

Uppgift 3 (6 p)

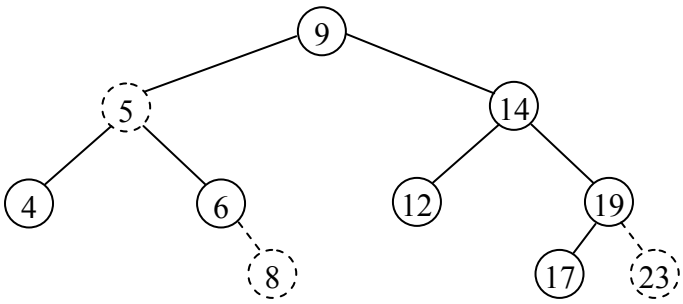
Insättningsföljderna 8;23;5, 8;5;23 samt 23;8;5 resulterar i trädet:



Om 5 sätts in före 8 bryts AVL-villkoret vid insättningen av 5 och 6 blir djupaste obalanserade noden. Fallet motsvarar fall 2 i Weiss och kräver en dubbelrotation . Efter rotation ser trädet ut så här:



Efter insättning av 8 och 23 i valfri ordning får vi



De sekvenser som ger ovanstående träd är 5;8;23, 5;23;8 samt 23;5;8.

Uppgift 4 (6 p)

Rekursivt

```
public static ListNode insert(int x,ListNode l) {
    if ( l == null || x <= l.element )
        return new ListNode(x,l);
    else {
        l.next = insert(x,l.next);
        return l;
    }
}
```

Iterativt

```
public static ListNode insert(int x,ListNode head) {
    ListNode p = head;
    while ( p.next != null && x > p.next.element )
        p = p.next;
    p.next = new ListNode(x,p.next);
    return head;
}
```

Uppgift 5 (5 p)

I en hashtabell där kvadratisk sondering används får fyllnadsgraden ej överstiga 0.5 och tabellstorleken skall vara ett primtal. Det garanterar att sonderingsalgoritmen terminerar med en ledig plats vid insättning av ett nytt element.

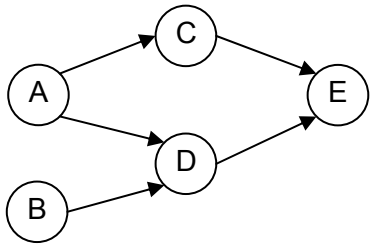
Den minsta lämpliga tabellstorleken för sex element är således 13. Så här ser tabellen ut efter anropssekvensen i uppgiften. Observera att platserna för de borttagna elementen 3 och 20 inte får användas för insättning av andra element med samma hashvärden.

0	
1	
2	
3	29
4	3
5	4
6	42
7	20
8	
9	
10	
11	
12	51

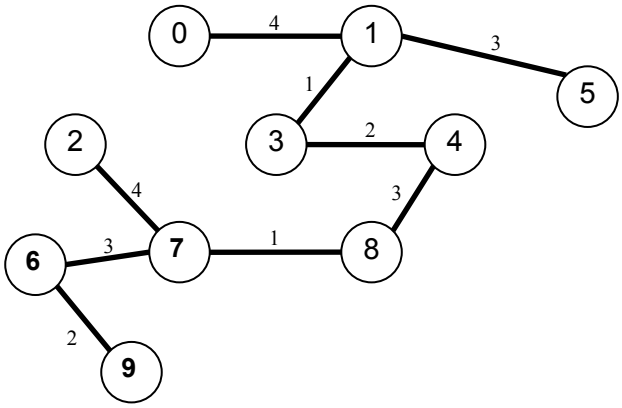
element	hashvärde
20	7
29	3
3	3
51	12
4	4
42	3

Uppgift 6 (5+6 p)

a) T.ex.



b)



Uppgift 7 (4+8 p)

a)

```
public static TreeNode heapAsTree(int[] a) {
    return heapAsTree(a,1);
}
private static TreeNode heapAsTree(int[] a,int root) {
    if ( root > a.length - 1 )
        return null;
    else
        return new TreeNode(a[root],
                            heapAsTree(a,2*root),
                            heapAsTree(a,2*root+1));
}
```

b)

```
public static int[] treeAsHeap(TreeNode root) {
    if ( root == null || ! isComplete(root) )
        throw new IllegalArgumentException("treeAsHeap applied to
                                            uncomplete tree");
    else {
        int[] a = new int[TreeNode.size(root)+1];
        a[0] = -9999999; // Sentinel
        Queue<TreeNode> q = new LinkedList<TreeNode>();
        q.add(root);
        int i = 1;
        while ( ! q.isEmpty() ) {
            TreeNode t = q.poll();
            a[i++] = t.element;
            if ( t.left != null )
                q.add(t.left);
            if ( t.right != null )
                q.add(t.right);
        }
        return a;
    }
}
```