Lösningsförslag till tentamen

Kursnamn	**Algoritmer och datastrukturer**
Tentamensdatum	**2016-06-03**
Program	**DAI2+I2**
Läsår	**2015/2016, lp 4**
Examinator	**Uno Holmer**

Uppgift 1 (10 p) Ingen lösning ges. Se kurslitteraturen.

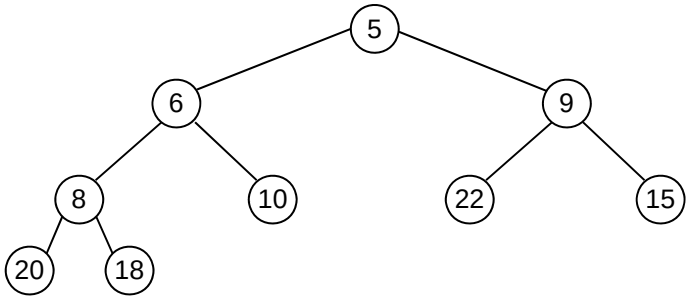
Uppgift 2 (5p eller 12 p)

a)
```
public String toString() {
    if ( isVar() )
        return "" + getVar();
    else if ( isConst() )
       return "" + getConst();
    else {
        String s1 = left.toString();
        String s2 = right.toString();
        return "(" + s1 + getOp() + s2 + ")";
    }
}
```
b) Denna lösning bortser från att `e` kan vara `null` (det borde gett undantag i konstruktorn).
```
public static Expression simplify(Expression e) {
    if ( e.isVar() || e.isConst() )
        return e;
    else {
        Expression e1 = simplify(e.getLeft());
        Expression e2 = simplify(e.getRight());
        if ( e.isPlus() ) {
            if ( e1.isConst() && e2.isConst() )
                return new Expression(e1.getConst() + e2.getConst());
            if ( e1.isConst() && e1.getConst() == 0 )
                return e2;
            if ( e2.isConst() && e2.getConst() == 0 )
                return e1;
            return new Expression('+',e1,e2);
        } else if ( e.isMult() ) {
            if ( e1.isConst() && e2.isConst() )
                return new Expression(e1.getConst() * e2.getConst());
            if ( e1.isConst() && e1.getConst() == 1 )
                    return e2;
            if ( e1.isConst() && e1.getConst() == 0 )
                    return new Expression(0);
            if ( e2.isConst() && e2.getConst() == 1 )
                    return e1;
            if ( e2.isConst() && e2.getConst() == 0 )
                    return new Expression(0);
            return new Expression('*',e1,e2);
        } else return e; // Impossible case
}}
```

Uppgift 3 (3+3 p)

a)

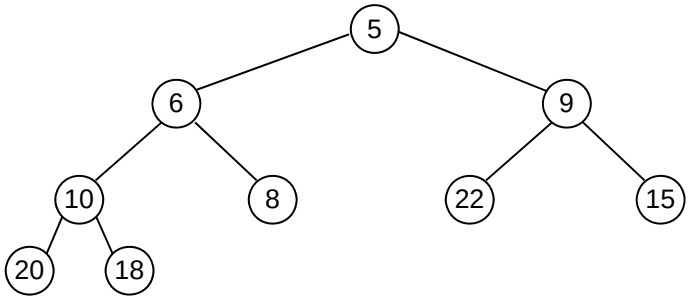
-∞	5	6	9	8	10	22	15	20	18
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*



b) När buildHeap utförs på fältet 10,20,15,5,8,22,9,6,18

-∞	10	20	15	5	8	22	9	6	18
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

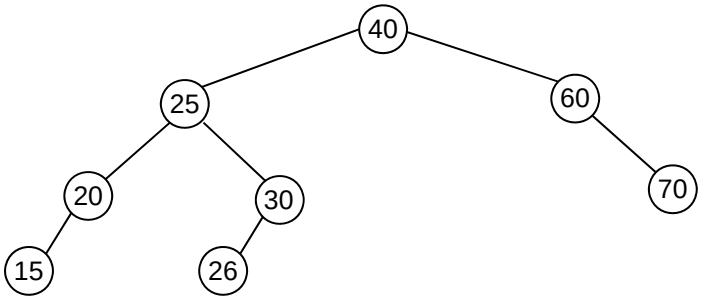
fås trädet



Talen 8 och 10 hamnar alltså inte på samma platser som efter insättningssekvensen med `insert`.

Uppgift 4 (3+3 p)

a) Vi får följande AVL-träd, vid insättning av 40 krävs en dubbelrotation enligt fall 3 i Weiss, och därefter en dubbelrotation vid insättning av 25 enligt fall 2 i Weiss:



b) Inorder: L H D I B J E A K F C G, preorder: A B D H L I E J C F K G,
postorder: L H I D J E B K F G C A.

Uppgift 5 (2+5+1 p)

a) I en effektiv implementering av Dijkstras algoritm behövs en optimalt implementerad prioritetskö. Dijsktras algoritm kan inte (i sin grundversion) hantera grafer med negativa kostnadscykler.

b) ABCDE, ABDCE, BACDE, BADCE, BCADE, BCDAE, BCDEA, BDACE, BDCAE, BDCEA.

c) En graf med N noder och inga bågar har $O(N!)$ topologiska ordningar. Eftersom alla noder har ingrad 0 kan man starta med vilken som helst och fortsätta med vilken som helst av de återstående, o.s.v. N noder kan permuteras på N! olika sätt.

Uppgift 6 (4+4 p)

a)

Talen sätts in på platserna (raderade tal är överstrukna):

0	~~10~~
1	
2	~~24~~
3	
4	
5	
6	
7	31
8	
9	~~9~~
10	21

b)

Tabellen blir mer än halvfull efter insättning av 13 (på plats 3). Eftersom tre element strukits har vi nu tre element att kopiera. Vi hashar därför om till en tabell med storlek minsta primtal större än 4*3 vilket är 13. (Vi väljer naturligtvis inte storleken 7 eftersom λ då blir nära 0.5 direkt.)

0	13
1	
2	
3	
4	
5	31
6	
7	
8	21
9	
10	
11	
12	

Uppgift 7 (10 p)

```
public static Consumer computeNetwork(Collection<Integer> demands) {
    if ( demands == null || demands.size() == 0 )
        throw new IllegalArgumentException();

    PriorityQueue<Consumer> pq = new PriorityQueue<>();
    for ( int demand : demands )
        pq.add(new Plant(demand));

    while ( pq.size() > 1 )
        pq.add(new Split(pq.poll(),pq.poll()));

    return pq.poll();
}
```