Anti-AVL

AVL-trær bruker venstre- og høyrerotasjoner for å sørge for at det binære søketreet ender opp balansert etter innsetting og sletting. Her er innsetting for en variant av AVL-trær som vi kaller *Anti-AVL*.

```
   Input: Rotnoden v av et Anti-AVL-tre B og et element x
   Output: Et Anti-AVL-tre med x satt inn
1  Procedure InsertAntiAVL(v, x)
2      if v = null then
3          v ← new Node(x)
4      else if x < v.element then
5          v.left ← InsertAntiAVL(v.left, x)
6      else if x > v.element then
7          v.right ← InsertAntiAVL(v.right, x)
8
9      if v.left ≠ null then
10         return RightRotate(v)
11
12     return v
```

```
   Input: En node z
   Output: Roten til det høyreroterte treet
1  Procedure RightRotate(z)
2      y ← z.left
3      T2 ← y.right
4
5      y.right ← z
6      z.left ← T2
7
8      return y
```

(a) Sett inn tallene $3, 1, 4, 2, 5$ med `InsertAntiAVL`.

- Beskriv kort hvordan det resulterende Anti-AVL-treet ser ut.
- Angi det totale antallet rotasjoner som forekommer i løpet av innsettingene.

(b) Beskriv mer generelt hvordan Anti-AVL-trær ser ut.

(c) Et binært søketre er et binærtre der elementet i hver node er:

- større enn alle elementer i venstre subtre, og
- mindre enn alle elementer i høyre subtre.

Har Anti-AVL-trær den samme egenskapen? Forklar.

(d) Hva er kjøretidskompleksiteten på oppslag i Anti-AVL-trær? Du kan anta at oppslag er implementert identisk som ordinære binære søketrær og AVL-trær. Begrunn svaret kort.

(e) Beskriv kort hvordan innsetting i Anti-AVL-trær skiller seg fra innsetting i AVL-trær.