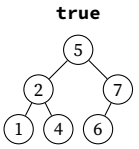
Er binærtreet et søketre? 10 poeng

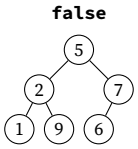
Anta at du er gitt et binærtre B med unike heltall. Hvis v er en node i det binære treet, så gir

- v.`element` heltallet som er lagret i noden
- v.`left` venstre barn av v
- v.`right` høyre barn av v

Vi ønsker å sjekke om det binære treet også er et binært *søketre*. Under finner du spesifikasjonen for algoritmen, og to eksempler på trær som henholdsvis bør gi **`true`** og **`false`**.

```
Input: Rotnoden v av et binærtre B
Output: Returnerer true hvis binærtreet er et binært
        søketre, false ellers
1 Procedure CheckBST(v)
  | // ···
```





Det finnes flere gode løsninger på dette problemet. Følgende kan være behjelpelig når du tenker på en løsning:

- Du kan anta at du har prosedyrer `FindMin` og `FindMax` som henholdsvis finner minste og største tall i det binære treet. Siden treet ikke er garantert å være et binært søketre (eller balansert) så vil disse prosedyrene ha lineær tid.
- Det kan være lurt å dele algoritmen opp ved å lage en hjelpeprosedyre.

(a) Skriv ned egenskapen et binærtre må ha for å kunne kalles et binært *søketre*.

(b) Fullfør prosedyren over. Lavere kjøretidskompleksitet er mer poenggivende.

(c) Oppgi kjøretidskompleksiteten på algoritmen din med hensyn til antall noder i binærtreet.

(a) Verdien i hver node må være *større enn alle* verdier i venstre subtre og *mindre enn alle* verdier i høyre subtre.

Det gis 2 poeng for å oppgi riktig egenskap. Spørsmålet er gitt mest for å sørge for at studenten har dette klart for seg før neste deloppgave.

(b) Her er tre «naturlige» løsninger på problemet.

En $\mathcal{O}(n^2)$ løsning som følger definisjonen av binære søketre godt kan se slik ut:

```
1  Procedure CheckBST(v)
2      if v = null then
3          return true
4      if v.element < FindMax(v.left) then
5          return false
6      if v.element > FindMin(v.right) then
7          return false
8      return CheckBST(v.left) and CheckBST(v.right)
```

Ved å regne ut minste og største noder i treet først får vi en fin $\mathcal{O}(n)$ løsning:

```
1  Procedure CheckBST(v)
2      return CheckBSTHelper(v, FindMin(v), FindMax(v))
3
4  Procedure CheckBSTHelper(v, low, high)
5      if v = null then
6          return true
7
8      if v.element < low or v.element > high then
9          return false
10
11     checkLeft ← CheckBSTHelper(v.left, low, v.element)
12     checkRight ← CheckBSTHelper(v.right, v.element, high)
13
14     return checkLeft and checkRight
```

En annen $\mathcal{O}(n)$ løsning er å gjøre en in-order traversering av det binære treet, og legge nodene i en liste. Dersom den resulterende listen er sortert, så er treet også et binært søketre.

```
1  Procedure CheckBST(v)
2      inorder_list ← InOrder(v, empty list)
3      for consecutive pairs (x, y) in inorder_list do
4          if x > y then
5              return false
6      return true
7
8  Procedure InOrder(v, list)
9      if v = null then
10         return list
11
12     InOrder(v.left, list)
13     append v.element to list
14     InOrder(v.right, list)
15
16     return list
```

Det gis opp til 4 poeng for en kvadratisk løsning, og opp til 6 poeng for en lineær løsning.

(c) Det gis 2 poeng for riktig kjøretidskompleksitet.