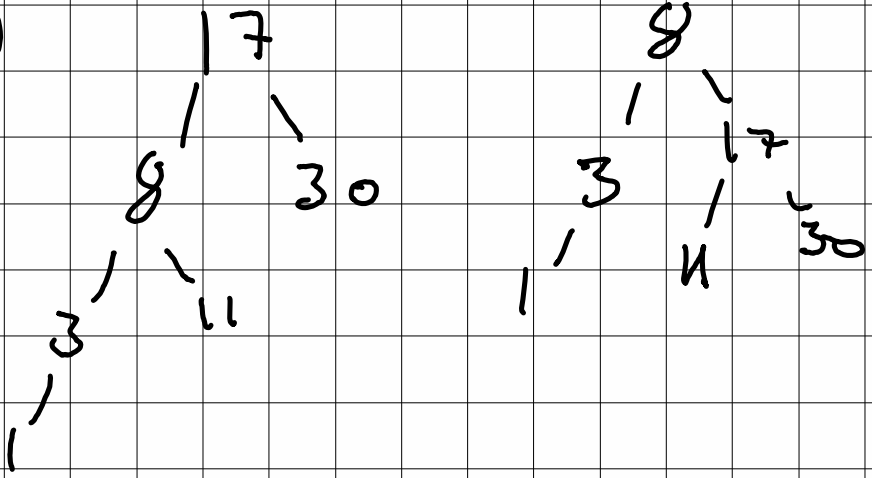
Oppg. 1

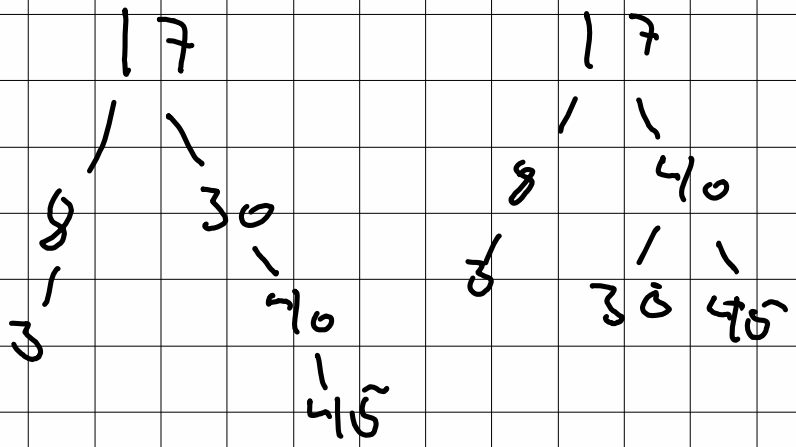
a) Eit Bs tre er balansert dersom det er fylt opp fra venstre og høgdeforskjellen mellom to høyre i høgre er mer enn. 1

b)



Gamle treet var ubalansert fordi høgden av venstre undertre er meir enn 1 høgare enn høgre undertre.

c)



Bs treet var ikke balansert
Siden høyden av venstre undertre er lik 0 mens høyden av høyre undertre er 2.

d) Det er viktig at treet er balansert fordi operasjoner på et Bs tre som er balansert er $O(\log n)$ som er veldig raskt.

Oppg. 2

a)

c e f b d a

b) c e b d a f

c) $G = (V, E)$

$V = (a, b, c, d, e, f)$

$E = ((a,d), (b,d), (b,e), (b,f), (f,c), (d,f), (e,f), (e,c))$

	a	b	c	d	e	f
a	0	0	0	1	0	0
b	0	0	0	1	1	1
c	0	0	0	0	1	1

a d

d, a, e, b

b, d, e, f

e, d, b, f, c

f, e, b, c

c, e, f

Oppgave 3.

kriterier for en god hashCode

hvis en klasse overskriver equals metoden bør den også overskrive hashCode

hvis equals metoden ser på to elementer som like bør hashkode gi samme verd

hvis du kaller eit Objekts hashCode fleire ganger i eit program og Objektets data forblir uendra skal den returnere samme kode

Eit Objekts hashCode i eit run av eit program kan være forskjellig fra eit anna Run.

Ein perfekt hashcode hadde krevd at ulike Objekt gav ulik hashCode. Men dette er veldig vanskelig å få til. Men en god hashCode skal unngå dette så masse som mulig.

Ein hashfunksjon produserer ein tabellindex utifrå sin entry sin Search key.

Ein kollisjon forekommer når ein hashCode gir samme Index til to forskjellige Objekter.

Kollisjoner som er løyst via linear Probing forårsaker grupper av etterfølgende lokasjoner i hashTabbet som er opptatt. Kvar gruppe kalles en Cluster.

Tab[1] = EL654

Tab[4] = TA143

Siden Tab[1] er ledig sjekker vi for Tab[1+1].

Tab[2] = ZX871

Tab[7] = EL4700

Tab[0] = VV5000

Siden Tab[4] ikke er ledig sjekker vi for Tab[4+1]

Tab[5] = VV1454

Tab[6] = EL3294

Tab[1] = EL65431 - 2x87181

Tab[4] = TA14374 - UV14544 - EL3x44

Tab[7] = EL47007

Tab[0] = VV50000

c) $S[0] \cdot 31^{(n-1)} + S[1] \cdot 31^{(n-2)}$

S[i] = er den iende karakteren i strengen

n = lengden av strengen

UniCode a = 97
UniCode b = 98

hashCode = $97 \cdot 31^{(2-1)} +$

$98 \cdot 31^{(2-2)}$

= 3105

"123"

UniCode 1 = 49

UniCode 2 = 50

UniCode 3 = 51

hashCode =

$49 \cdot 31^{(3-1)} + 50 \cdot 31^{(2-1)} + 51 \cdot 31^{(1-1)} = \underline{\underline{48690}}$

d) Grunnen til at dei to utskri-ftene ikkje gir same resultat er fordi med mindre man overskriver hashCode metoden i Student klassen vil den bruke den generelle metoden til Objekt for å lage en int, denne metoden bruker minne

adressen til et Objekt for å gjøre dette. Så selv om a og b er like vil den ikke returnere samme hashCode fordi de ikke er adressert til samme sted i minne.

For å fikse dette må vi skrive vår egen hashCode metode:

c) Når vi sjekker for elementer i hashset og Tabellen vår via binarySearch. Ser me at hashset er raskere enn binarySearch.
Siden vi garanterer at det ikke er noen duplikater vil effektiviteten til Contains metoden bli: O(1)

Mens effektiviteten til
binarySearch er

Worst Case $O(\log(n))$

Best Case $O(1)$

Average Case $\Theta(\log(n))$

Siden Best Case er veldig
sjelden (Skjer når elementet
befinner seg i midten av
tabellen) vil HashSet naturligvis
være raskere enn BinarySearch.