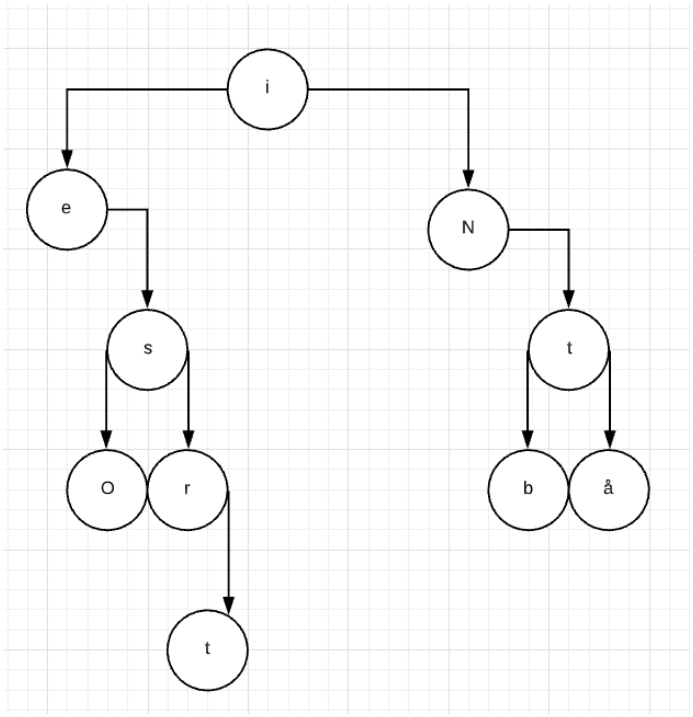
Øving 7 DAT102 Joachim Leiros

Oppgave 1

a) Ett binært tre er en datastruktur bestående av noder som har enten 0, 1 eller 2 barn. Høyden på ett binært tre er antall nivåer med noder i treet. Diverse dersom node A har node B som barn og B har node C som barn er høyden = 3.
Ett fult binært tre er ett binært tre hvor alle nivåene er fyllt opp med noder. Diverse høyde 1 kan ha 1 node, høyde 2 har 1 + 2 = 3 noder, høyde 3 har 1 + 2 + 4 = 7 noder osv.
Ett komplett binært tre er ett binært tre hvor alle nivåer bortsett fra det laveste er komplette. I det laveste skal alle nodene ligge så langt til venstre i treet som mulig.

b)

						i					
		E							n		
	S			t				O		r	
b		å	t								



c)
i)
pre: i E S t b å t n O r
in: b S å E t t i O n R
post: b å t s t e O r n i
nivå: i E n S t O r b å t
ii)
pre: i e s O r t N t b å
in: e O s r t I N b t å
post: O t r s e b å t n i
nivå: i E n S t O r b å t

Oppgave 2)
Vedlagt kode
b)

```
/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java ...
Antall noder: 1024
Teoretisk minimumshøyde: 10
Teoretisk maksimumshøyde: 1023
MIN høyde: 18
MAX høyde: 27
Gjennomsnittshøyde: 21

Process finished with exit code 0
```

c)

```
/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java ...
Antall noder: 8192
Teoretisk minimumshøyde: 13
Teoretisk maksimumshøyde: 8191
MIN høyde: 26
MAX høyde: 37
Gjennomsnittshøyde: 29

Process finished with exit code 0
```

Oppgave 3

a) En binær haug er ett binært tre hvor etthvert barne element som regel har lavere/høyere eller lik verdi i forhold til roten, slik at det blir lettere å lete i treet. En binær haug er også alltid komplett, diverse alle nivåer bortsett fra det laveste må være fyllt opp i alle noder. Diverse ingen noder i nivå 3 kan ha kun 1 barn før vi lager nivå 4.

b)

A – Ikke en makshaug da 9 <= 15.

B- Makshaug ettersom alle noder >= barn.

						15					
		12							10		
	11			2				6		3	
4		8	1								

C – Makshaug ettersom alle noder >= barn.

						15					
		10							14		
	8			7				13		6	
2		5	4								

c)

						14					
		12							6		
	11			10				2		5	
1		7	4								

Etter fjerning av 14:

						12					
		11							6		
	7			10				2		5	
1		4	4								

Setter inn 13:

						13					
		12							6		
	7			11				2		5	
1		4	4		10						

d)

Metoden kan brukes til å sortere elementer etter størrelse da hvert nivå vil bestå av elementer som er lavere enn det forrige nivået. Noe som gjør det lett anvendelig til søking da man lett kan finne fram til en verdi ved å gå nedover i treet, framfor å måtte søke i bredden.

e)

					10				
		9					8		
	7			6		5		4	
3		2	1						

					1				
		9					8		
	7			6		5		4	
3		2	10						

					1				
		2					8		
	7			6		5		4	
3		9	10						

					1				
		2					8		
	3			6		5		4	
7		9	10						

					1				
		2					4		
	3			6		5		8	
7		9	10						

f)
Utskrift av kode:

```
/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java ...
Verdier i tabellen er:
1 10 2 18 54 33 30 300 200 100

Haugen i sortert rekkefølge:
1  2  10  18  30  33  54  100  200  300

Process finished with exit code 0
```

Oppgave 4

a) Ett 2-3 tre er ett binært søketre hvor hver node har null, to eller tre barn. Ett 2-3 tre brukes til å redusere antall nivåer i det lagrede treet, og redusere mengden data som blir overført når man overfører data fra disk til minne. I tillegg reduserer vi høyden på treet ved at alle noder har 0, 2 eller 3 barn. Altså ingen noder med kun 1 barn.

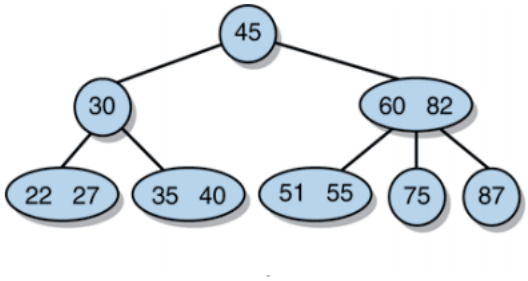
b)
En 2-node er en node som har 0 eller 2 barn. Elementene i venstre undertre har mindre verdi enn elementet i 2-noden, elementene i høyre undertre har større eller lik verdi i forhold til mengden i 2-noden.

2-node		10		
5			7	
3	6		4	7

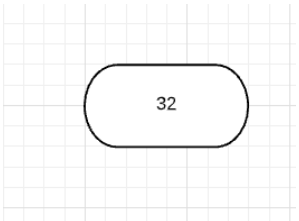
En 3-node er en node som inneholder 2 elementer og har 0 eller 3 barn hvor elementer i venstre undertre er mindre enn det minste av elementene. Elementene i det midterste undertreet har en verdi som er større enn det minste elementet og mindre enn det største elementet. Div. Minste <= midterste <= største.
Det høyre undertreet består av elementer som er større eller lik den største mengden i 3-noden.

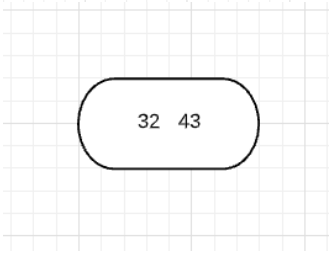
3-node			10			
5 && 8				7 && 9		
3	6	8		4	7	10

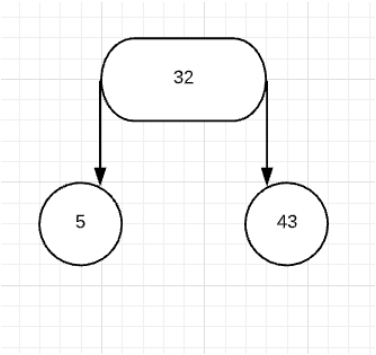
c)

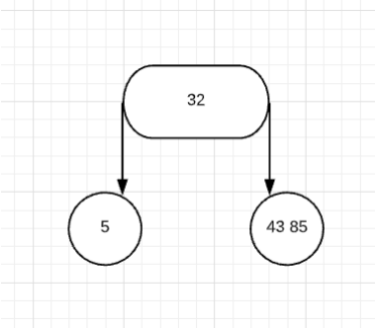


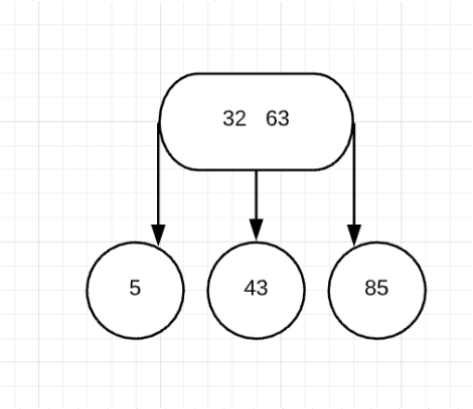
Vi starter på 45 og siden 42 < 45 så går vi inn i venstre undertre. 42 > 30 så går vi videre i høyre undertre. 45 > 40 så leter vi på høyre undertre til 35-40 noden, men siden vi ikke finner verdien der så finnes ikke verdien i treet.

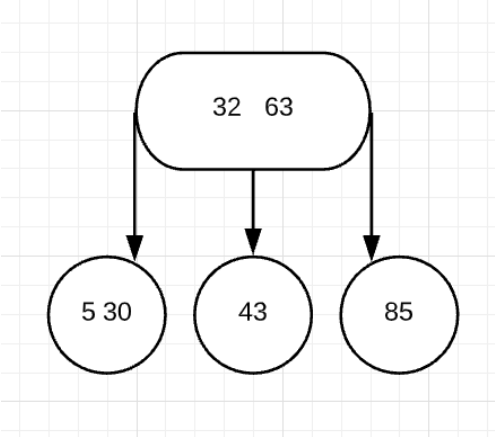


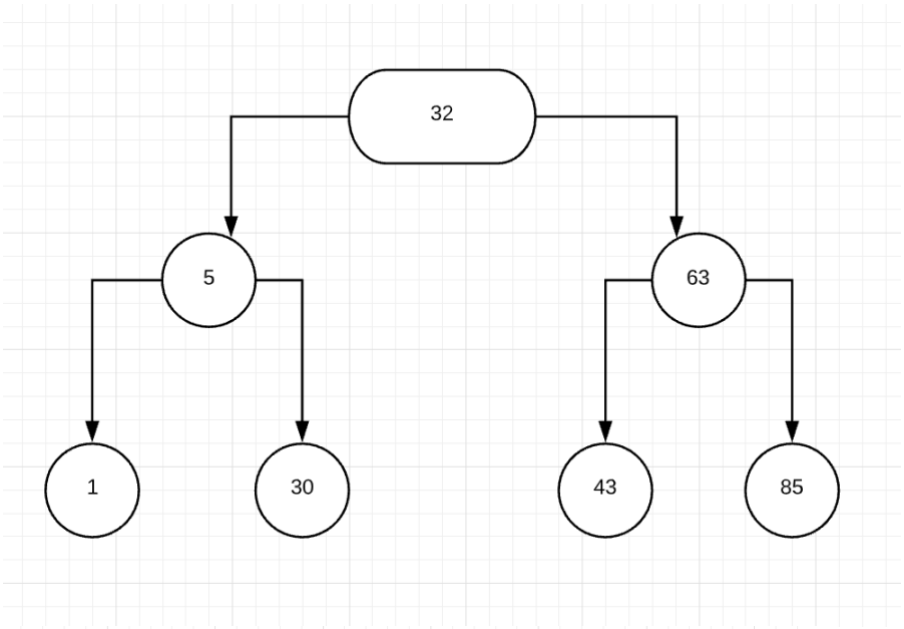


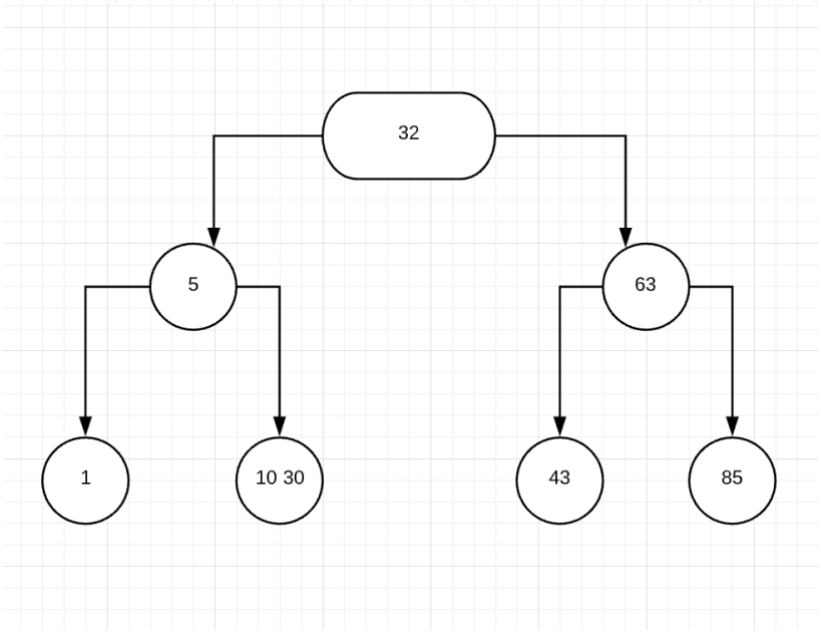

32
5
43

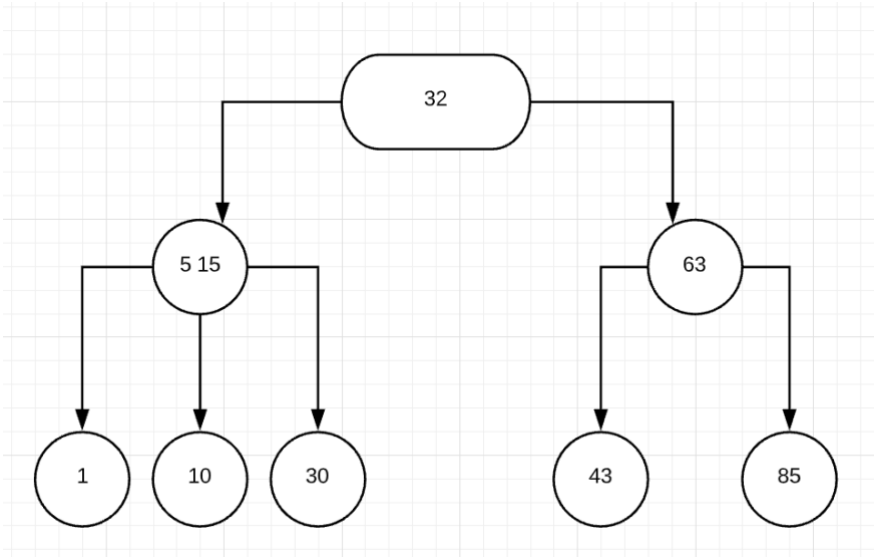

32
5
43 85


32 63
5
43
85


32 63
5 30
43
85


32
5
63
1
30
43
85


32
5
63
1
10 30
43
85


32
5 15
63
1
10
30
43
85

Oppgave 5

a) Bredde-først

Stabel	Besøkt
c	c
ef	ce
fbd	cef
bd	cefb
d	cefbd
a	cefdba
tom	

b) Dybde først

Stabel	Besøkt
c	c
e	ce
b	ceb
d	cebd
a	cebda
dae	cebda
bef	cebda
fcbe	cebdaf
ecdf	cebdaf
cf	cebdaf
tom	

c)
V = [a,b,c,d,e,f]
E = [(c,f) , (c,e) , (f,b) , (f,e) , (e,d) , (b,d) , (d,a) , (b,e)]

Nabomatrise og naboliste:

#	a	b	c	d	e	f
A				X		
B				X	X	X
C					X	X
D	X	X			X	
E		X	X	X		X
F		X	X		X	

(a)0	D
(b)1	D,e,f
(c)2	E,f
(d)3	A,b,e
(e)4	C,f,b,d
(f)5	B,c,e

Oppgave 6

a) I Hashing er elementene lagret i en hashtabell, posisjonene til alle elementer er ideelt set lagret i hashverdien. Altså alle mengder har en bestemt adresse i hashtabellen som gjør det raskt og hente fram.
Hash-funksjon: En funksjon som brukes til å kartlegge data i en hashtaball.
Kollisjon: En situasjon hvor to ulike hash nøkler peker til samme element i en celle.
Clustering: En situasjon som oppstår etter en hash kollisjon som fører til at elementet fra den ene hash nøkkelen i kollisjonen blir flyttet til neste søkeadresse.

En god hash-funksjon har hashverdien bestemt av elementene i hashtabellen. Funksjonen bruker all data, lagrer elementene jevnt i tabellen uten store tomrom og funksjonen klarer å genererer ulike hashverdier for liknende hashverdier.

b)

c)
ab = 97*31+98*1 = 3105

123 = (49*31^2) + (50*31^1) + 51 = 48690

```
/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java ...
ab - HashCode: 3105
123 - HashCode: 48690

Process finished with exit code 0
```

d) Når vi benytter oss av equals metoden må vi også overkjøre hashkoden slik at like objekt får lik hashkode.

e)

```
/Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java ...
HashSet
Tid: 1ms.
Antall treff: 1025
Integer array
Tid: 34ms.
Antall treff: 1025

Process finished with exit code 0
```

Vi finner naturligvis likt antall treff i tabellen men hashset funksjonen er særdeles mye raskere enn binærsøking da vi kan lete oss direkte fram til elementet ved hjelp av hash-nøkkelen.