Høgskolen i Østfold

EKSAMEN

Emnekode: **ITF20006**	Emne: **Algoritmer og datastrukturer**
Dato: **28. mai 2018**	Eksamenstid: **09:00 – 13:00**
Hjelpemidler: **Alle trykte og skrevne**	Faglærer: **Jan Høiberg**

Om eksamensoppgavene:

Oppgavesettet består av 12 sider, inkludert denne forsiden og 1 vedlegg. Kontroller at oppgavesettet er komplett før du begynner å besvare spørsmålene.

Oppgavesettet har 4 oppgaver, hver av dem har flere deloppgaver. Innen hver oppgave vektes alle deloppgaver likt. Les oppgavetekstene nøye før du begynner på besvarelsen.

Legg vekt på å skrive en ryddig og lett forståelig besvarelse.

Sensurfrist: **18. juni 2018**

Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist.
www.hiof.no/studentweb

Oppgave 1: Flervalgsoppgave (30%)

Denne oppgaven har 15 delspørsmål. Alle spørsmålene har to, tre eller fire svaralternativer (nummerert fra a til b, c eller d), men bare ett av dem er riktig. Du skal for hvert spørsmål angi hvilket av alternativene *du* mener er det riktige.

Skriv svarene dine på denne formen[1]:

```
Spørsmål:  1  2  3  4 ...   14 15
Svar:      a  b  c  d        b  a
```

Spørsmål:

1. Hvilken av følgende datastrukturer lagrer alltid dataene i *sortert* rekkefølge?

a) Stack
b) Lenket liste
c) Heap
d) Ingen av alternativene a – c ovenfor er sorterte.

2. Hvilken av følgende datastrukturer lagrer alltid dataene i *sortert* rekkefølge?

a) Binært søketre
b) AVL-tre
c) B-tre
d) Alle alternativene a – c ovenfor er sorterte.

3. Hvilken av følgende datastrukturer er mest effektiv å bruke til å implementere en prioritetskø som skal lagre et stort antall elementer?

a) Stack
b) Kø
c) Heap
d) Hashtabell

4. Kjøretiden til flettesortering (merge sort) av en array avhenger av:

a) Lengden på arrayen som skal sorteres.
b) Verdiene og rekkefølgen på dataene som skal sorteres.
c) Antall like verdier i arrayen.
d) Alle de tre faktorene som er nevnt i alternativene a – c ovenfor.

(Oppgave 1 fortsetter på neste side)

1 Svarene angitt her er bare eksempler og ikke nødvendigvis riktige.

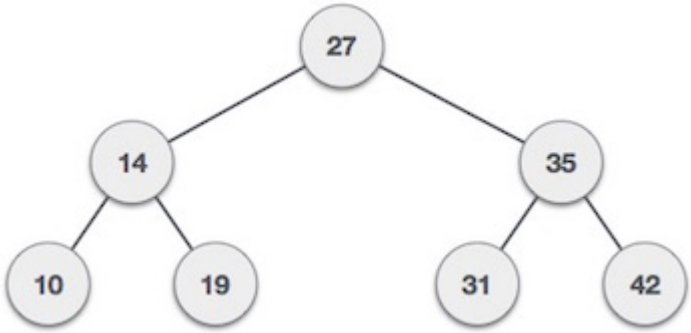
5. Hvilken av følgende sorteringsmetoder kan *ikke* garantere at arbeidsmengden *alltid* er *O(n* log *n):*

a) Merge sort
b) Quicksort
c) Heapsort
d) Alle alternativene a) – c) ovenfor har arbeidsmengder som garantert er *O(n* log *n).*

6. Hvilken av følgende sorteringsmetoder er raskest for svært lange arrayer med flere titalls millioner floating point tall, der dataene er tilfeldig (random) fordelt?

a) Shell sort
b) Heapsort
c) Quicksort
d) Merge sort

7. Følgende binære søketre skal traverseres:



Nodene oppsøkes i denne rekkefølgen: 27 14 35 10 19 31 42 . Hvilken traverseringsmetode er brukt?

a) Preorder
b) Inorder
c) Postorder
d) Bredde-først

8. Er det binære søketreet i spørsmål 7 også et AVL-tre?

a) Ja
b) Nei

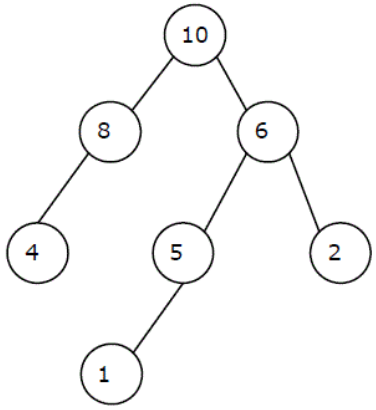
9. Er det binære søketreet i spørsmål 7 også et B-tre?

a) Ja
b) Nei

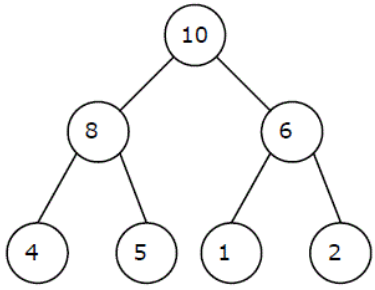
(Oppgave 1 fortsetter på neste side)

10. I en maks-heap er verdien til en foreldernode alltid større eller lik verdien i barna. Hvilken av disse fire figurene er en maks-heap?

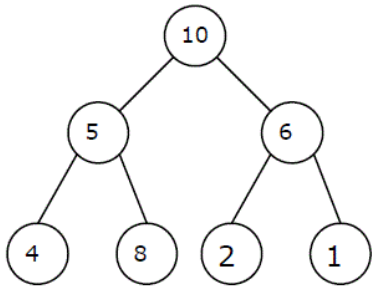
(A)



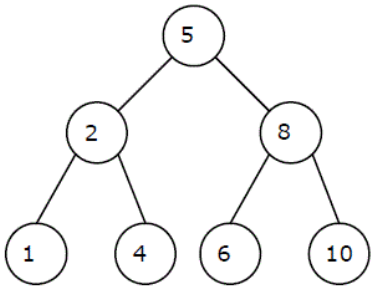
(B)



(C)



(D)



11. En heap implementeres vanligvis med bruk av en array. Hvilken av disse fire arrayene representerer en maks-heap?

a) `{ 25, 12, 16, 13, 10, 8, 14 }`

b) `{ 25, 14, 16, 13, 10, 8, 12 }`

c) `{ 25, 14, 16, 13, 10, 8, 17 }`

d) `{ 25, 13, 16, 12, 14, 8, 12 }`

12. Load factor L for en hashtabell er forholdet mellom antall dataelementer n som er lagret, og lengden H på hastabellen:

$$L = \frac{n}{H}$$

Under kjøring av et program som bruker hashing, ble verdien $L = 1.4$ beregnet. Hvilken teknikk ble brukt i programmet for å håndtere kollisjoner i hashtabellen?

a) Åpen adressering med lineær probing

b) Åpen adressering med kvadratisk probing

c) Hashing med kjeding

d) Rehashing

(Oppgave 1 fortsetter på neste side)

13. Figuren nedenfor viser en hashtabell med lengde lik 10, etter det er satt inn 6 heltall. Det brukes åpen adressering med lineær probing. Hashverdien *h* for en dataverdi *k* beregnes som resten ved heltallsdivisjon med hashlengden: *h(k)* = *k* mod 10. Dvs. at hashindeksen til et heltall alltid er lik det siste sifferet i tallet.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

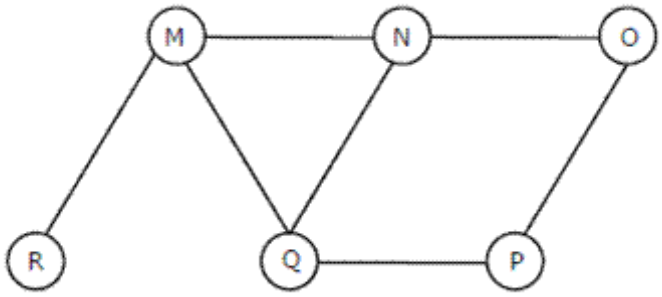
Hvilken av de fire rekkefølgene nedenfor er brukt ved innsetting av de 6 tallene?

a) 46 42 34 52 23 33
b) 34 42 23 52 33 46
c) 46 34 42 23 52 33
d) 42 46 33 23 34 52

14. Hvilken hjelpedatastruktur brukes i dybde-først traversering av en graf?

a) Heap
b) Kø
c) Hashtabell
d) Ingen av alternativene a – c ovenfor.

15. Følgende urettede graf er gitt:



Hvilken av disse fire rekkefølgene av noder er en (standard kø-basert) bredde-først traversering av grafen?

a) M N O P Q R
b) N Q M P O R
c) Q M N P R O
d) Q M N P O R

(Slutt på oppgave 1)

Oppgave 2: Stack og kø (15%)

I deloppgavene a) – c) nedenfor er det gitt tre Java-funksjoner som bruker enten en stack eller en kø. Dataene som lagres på stack eller i kø er enkle heltall. Alle funksjonene har en parameter `n` som er et positivt heltall.

Beskriv kort hva som *skrives ut* ved et kall på hver av funksjonene. Det er tilstrekkelig å angi hva som skrives ut for en gitt verdi av `n`, f.eks. for `n` = 10.

a)
```
void metode_a(int n)
{
    stack S = new stack();

    for (int i = 1; i <= n; i++)
        S.push(i);

    for (int i = 0; i < n; i++)
        System.out.print(S.pop() + " ");
    System.out.println();
}
```

b)
```
void metode_b(int n)
{
    queue Q = new queue();

    for (int i = 1; i <= n; i++)
        Q.enqueue(i);

    for (int i = 0; i < n; i++)
        System.out.print(Q.dequeue() + " ");
    System.out.println();
}
```

c)
```
void metode_c(int n)
{
    stack S = new stack();
    S.push(0);
    S.push(1);

    for (int i = 3; i <= n; i++)
    {
        int f2 = S.pop();
        int f1 = S.pop();
        S.push(f1);
        S.push(f2);
        S.push(f1 + f2);
    }

    for (int i = 1; i <= n; i++)
        System.out.print(S.pop() + " ");
    System.out.println();
}
```

(Oppgave 2 fortsetter på neste side)

d) Alle de tre funksjonene i deloppgavene a) – c) ovenfor har samme (asymptotiske) arbeidsmengde. Hva er arbeidsmengden til hver funksjon, angitt med *O*-notasjon? Gi en kort begrunnelse for svaret.

(Slutt på oppgave 2)

Oppgave 3: B-trær (20%)

I denne oppgaven skal det brukes B-trær av orden 4. Dataene som skal lagres i B-trærne består bare av heltall.

a) I et B-tre av orden 4:

i) Hva er minste og største antall verdier som kan lagres i en node?

ii) Hva er minste og største antall barn som en indre node i treet kan ha?

Når en node i B-treet ikke har plass til en ny verdi, skal noden deles på følgende måte:

- Den minste verdien i noden skal legges i en ny node til venstre.
- De to største verdiene i noden skal legges i en ny node til høyre.
- Den nest minste verdien skal sendes opp til nivået ovenfor i treet.

Følgende 12 verdier skal settes inn i B-tre av orden 4, som i utgangspunkt er *tomt*, i denne gitte rekkefølgen:

16 4 8 12 7 11 15 6 7 2 3 5

b) Tegn en figur som viser hvordan B-treet ser ut etter at de tre første verdiene 16, 4, og 8 er satt inn.

c) Tegn en figur som viser hvordan B-treet du tegnet i deloppgave b) ser ut etter at de tre neste verdiene 12, 7, og 11 er satt inn.

d) Tegn en figur som viser hvordan B-treet du tegnet i deloppgave c) ser ut etter at de resterende verdiene 15, 6, 7, 2, 3 og 5 er satt inn.

e) Hva er høyden av dette B-treet etter at alle de 12 verdiene er satt inn?

(Slutt på oppgave 3)

Oppgave 4: Binære søketrær (35%)

I denne oppgaven skal det brukes vanlige binære søketrær (ikke AVL-trær), der hver node i treet bare inneholder et enkelt heltall i tillegg til pekere/referanser til nodens venstre og høyre barn.

Det er tillatt med flere like verdier i treet. Ved innlegging av en verdi som finnes i treet fra før, skal denne havne til *høyre* for en node med samme verdi.

Følgende 12 verdier skal settes inn i et søketre som i utgangspunkt er *tomt*, i denne gitte rekkefølgen:

16 4 8 12 7 11 15 6 7 2 3 5

a) Tegn en figur som viser hvordan treet ser ut etter at de 12 verdiene ovenfor er satt inn.

b) Hva er høyden på søketreet som du tegnet i deloppgave a) ovenfor?

c) Tegn en figur som viser hvordan treet du tegnet i deloppgave a) ovenfor ser ut etter at verdien 4 er fjernet fra treet.

I deloppgavene d), f) og g) nedenfor, skal du skrive eller skissere Java-metoder for binære søketrær. En ufullstendig implementasjon av binært søketre med enkle heltallsdata er gitt i klassen `intSearchTree` i vedlegg 1.

d) Klassen `intSearchTree` i vedlegg 1 inneholder en funksjon:

```
public int numOdd()
```

som skal beregne antallet oddetall (heltall som ikke er delelige med 2) som er lagret i søketreet. For f.eks. et søketre som bare inneholder de 12 verdiene som brukes i deloppgave a) ovenfor, vil denne funksjonen returnere verdien 6 fordi søketreet inneholder de 6 oddetallene 7, 11, 15, 7, 3 og 5.

Funksjonen `numOdd` kaller en intern hjelpefunksjon:

```
private int countOdd(treeNode r)
```

som gjør selve opptellingen av antall oddetallsverdier. Skriv funksjonen `countOdd` . Funksjonen *skal* være rekursiv.

e) Hva er arbeidsmengden for funksjonen `countOdd` , uttrykt med *O*-notasjon, for et søketre som inneholder *n* verdier?

(Opppgave 4 fortsetter på neste side)

f) Klassen `intSearchTree` i vedlegg 1 inneholder en funksjon:

```
public void print(int a, int b)
```

der verdien av parameteren `a` skal være mindre eller lik verdien av `b`. Funksjonen `print` skal skrive ut alle verdiene i søketreet som er større eller lik `a` og mindre eller lik `b`, i sortert rekkefølge. For f.eks. et søketre som bare inneholder de 12 verdiene som brukes i deloppgave a) ovenfor, skal funksjonskallet `print(4,9)` resultere i utskriften:

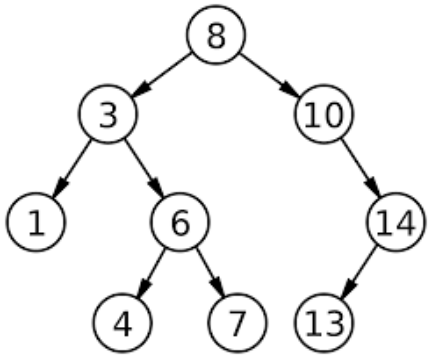
```
4 5 6 7 7 8
```

Funksjonen `print` kaller en intern hjelpefunksjon:

```
private void printRange(treeNode r, int a, int b)
```

som gjør selve utskriften. Skriv funksjonen `printRange`. Funksjonen *skal* være rekursiv og helst så effektiv som mulig.

g) I et binært tre er *hjørnenodene* definert som de nodene som ligger hhv. lengst til venstre og lengst til høyre på hvert nivå i treet. Det finnes enten en (hvis det bare er én enkelt node i et nivå) eller to hjørnenoder for hvert nivå i treet. I det binære treet i figuren nedenfor har hjørnenodene (fra øverste nivå og nedover i treet) verdiene 8, 3 og 10, 1 og 14, og til slutt 4 og 13.



Beskriv, enten med tekst eller kode, en algoritme for å skrive ut bare verdiene i alle hjørnenodene i et binært tre.

Hint: Ta utgangspunkt i algoritmen for bredde-først traversering av binære trær.

(Slutt på oppgave 4)

Vedlegg 1

Enkel og ufullstendig implementasjon av binært søketre med heltall

---

```
public class intSearchTree
{
    // Indre klasse for nodene i treet
    //
    private class treeNode
    {
        private int value;
        private treeNode left, right;

        private treeNode(int v)
        {
            value = v;
            left = right = null;
        }
    }

    private treeNode root;   // Roten i søketreet

    // Konstruktør for å opprette tomt søketre
    //
    public intSearchTree(){ root = null; }

    // Funksjon for å sjekke om treet er tomt
    //
    public boolean isEmpty() { return root == null; }

    // Innsetting av ny verdi i søketre
    //
    public void insert(int value)
    {
        if (root == null)
        {
            root = new treeNode(value);
            return;
        }

        treeNode current = root, parent = null;

        while (current != null)
        {
            parent = current;
            current = (value < parent.value) ? parent.left : parent.right;
        }

        current = new treeNode(value);

        if (value < parent.value)
            parent.left = current;
        else
            parent.right = current;
    }
```

```
    // Beregner antallet oddetall som er lagret i treet
    //
    public int numOdd()
    {
        return countOdd(root);
    }

    private int countOdd(treeNode r)
    {
        // Skal programmeres i oppgave 4 d)
    }

    // Skriver ut alle verdier x i treet som er slik at a <= x <= b
    //
    public void print(int a, int b)
    {
        printRange(root, a, b);
        System.out.println();
    }

    private void printRange(treeNode r, int a, int b)
    {
        // Skal programmeres i oppgave 4 f)
    }

} // Slutt på klassen intSearchTree
```