Høgskolen i Østfold

Ny/utsatt

EKSAMEN

Emnekode: **ITF20006**	Emne: **Algoritmer og datastrukturer**
Dato: **5. januar 2018**	Eksamenstid: **09:00 – 13:00**
Hjelpemidler: **Alle trykte og skrevne**	Faglærer: **Jan Høiberg**

Om eksamensoppgavene:

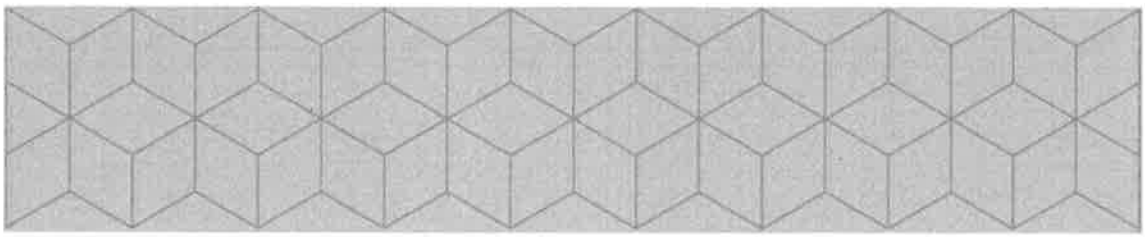
Oppgavesettet består av 12 sider, inkludert denne forsiden og 2 vedlegg. Kontrollér at oppgavesettet er komplett før du begynner å besvare spørsmålene.

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: **26. januar 2018**

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



Oppgave 1: Flervalgsoppgave (25%)

Denne oppgaven har 20 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 ...   19 20
Svar:      a  b  c  d        b  a
```

Spørsmål:

1. Følgende gjelder for en min-heap:
   a) Foreldernoder har verdier som er større eller lik verdiene i barna.
   b) Foreldernoder har verdier som er mindre eller lik verdiene i barna.
   c) Både alternativ a og alternativ b ovenfor er riktig.
   d) Både alternativ a og alternativ b ovenfor er feil.

2. Følgende gjelder for en max-heap:
   a) Den brukes til å lagre maksimumsverdier.
   b) Barnenoder har verdier som er ekte mindre enn verdien i foreldernoden.
   c) Barnenoder har verdier som er større enn verdien i foreldernoden.
   d) Den største verdien ligger lagret i rotnoden.

3. Hvilken hjelpe-datastruktur brukes i en bredde-først traversering av en graf?
   a) Stack
   b) Kø
   c) Hashtabell
   d) Ingen av alternativene a – c ovenfor

4. Hvilken hjelpedatastruktur brukes i bredde-først traversering av et søketre?
   a) Stack
   b) Kø
   c) Hashtabell
   d) Ingen av alternativene a – c ovenfor.

5. En lenket liste er en dynamisk datastruktur.
   a) Sant
   b) Usant

(Oppgave 1 fortsetter på neste side)

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

6. Kjøretiden til Quicksort avhenger:

a) Bare av antall verdier som skal sorteres.
b) Bare av metoden som brukes til å velge pivotelementer.
c) Bare av rekkefølgen på dataene som skal sorteres.
d) Av alle de tre faktorene som er nevnt i alternativene a – c ovenfor.

7. Balansefaktorene i et AVL-tre brukes til å sjekke:

a) Om treet skal få en ny rotnode etter innsetting av en verdi.
b) Om alle barnenoder er på samme nivå.
c) Når den siste rotasjonen i treet ble utført.
d) Om treet har blitt ubalansert ved innsetting eller fjerning av verdier.

8. Hvilken av følgende traverseringsmetoder skriver ut verdiene i et vanlig binært søketre i stigende sortert rekkefølge?

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

9. Hvilken av følgende datastrukturer lagrer dataene i sortert rekkefølge?

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

10. Hvilken av følgende søkemetoder er raskest (når arbeidsmengdene angis med O-notasjon) for svært store, sorterte arrays, der verdiene er relativt jevnt fordelt mellom minste og største verdi?

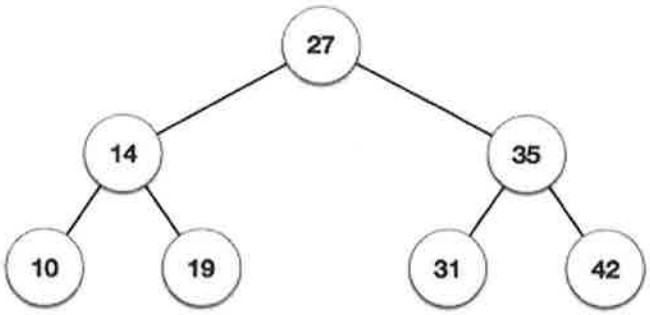
a) Binært søk
b) Ternært søk
c) Interpolasjonssøk
d) Sekvensielt søk

11. Hvilken sorteringsmetode egner seg best for arrayer som er nesten sorterte?

a) Bubble sort
b) Shell sort
c) Innstikksortering

(Oppgave 1 fortsetter på neste side)

12. Følgende *vanlige* binære søketre skal traverseres:



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

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

13. Verdien 15 skal settes inn i søketreet fra oppgave 12. Hvor havner den nye noden som inneholder 15?

a) Til høyre for 19
b) Til venstre for 19
c) Som ny rotnode i treet
d) Mellom 10 og 19

14. Verdien 35 skal fjernes fra søketreet i oppgave 12. Hvilken verdi vil ligge til høyre for rotnoden etter fjerningen?

a) 31
b) 27
c) 19
d) 14

15. Hvordan ser infix-regneuttrykket *(a + b) * (c + d)* ut når det skrives med postfix-notasjon?

*a) a b * c d + +*
*b) * + a b + c d*
c) a b + c d + *
d) Ingen av alternativene a – c ovenfor er riktige.

(Oppgave 1 fortsetter på neste side)

16. Hvilken av følgende datastrukturer lagrer dataene i sortert rekkefølge?

a) Binært søketre
b) B-tre
c) AVL-tre
d) Alle alternativene a – c ovenfor lagrer dataene sortert.

17. Effektiviteten til en hashtabell avhenger av:

a) Hashfunksjonen
b) Hashlengden
c) Load factor
d) Alle alternativene a – c ovenfor .

18. En vanlig kø implementeres mest effektivt med bruk av:

a) En array der starten på køen alltid ligger først i arrayen.
b) En array der slutten på køen alltid ligger først i arrayen.
c) En hashtabell
d) En sirkulær array

19. En prioritetskø implementeres mest effektivt med bruk av:

a) En heap
b) En hashtabell
c) Et AVL-tre
d) En sirkulær array

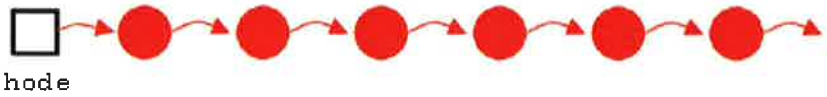
20. Hvilken sorteringsmetode har *alltid* en arbeidsmengde som er $O(n \log n)$ for sortering av en array av lengde n?

a) Shell sort
b) Heapsort
c) Quick sort
d) Radixsortering

(Slutt på oppgave 1)

Oppgave 2: Kø (25%)

En *toveiskø* er en kø der en både kan ta ut og legge inn verdier i begge ender. I denne oppgaven skal det lages en toveiskø ved hjelp en enveis pekerkjede som kun har en hodepeker til første element i køen:



I vedlegg 1 er det gitt en ufullstendig klasse `twowayQueue`, som implementerer en toveiskø med en slik enkeltlenket liste. Dataene i hver node i køen er bare et enkelt heltall.

Du skal bruke denne klassen i de to deloppgavene nedenfor. Det skal *ikke* legges inn flere instansvariabler eller statiske variabler i klassen `twowayQueue` eller i den indre klassen `node`.

a) Lag disse to metodene i klassen `twowayQueue`:

```
public void addFirst(int value)
public int removeFirst()
```

Den første metoden skal legge en ny node med angitt verdi først i køen. Den andre metoden skal fjerne den første noden og returnere nodens verdi. Du trenger ikke å skrive kode for å håndtere feilsituasjoner, som f.eks. fjerning fra tom kø.

b) Lag disse to metodene i klassen `twowayQueue`:

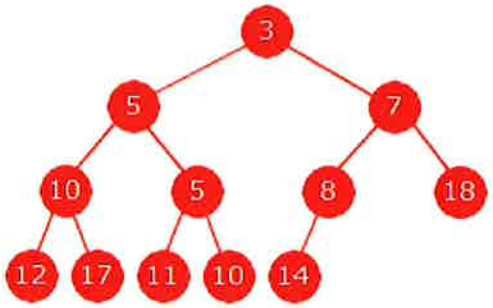
```
public void addLast(int value)
public int removeLast()
```

Den første metoden skal legge en ny node med angitt verdi sist i køen. Den andre metoden skal fjerne den siste noden og returnere nodens verdi. Du trenger ikke å skrive kode for å håndtere feilsituasjoner, som f.eks. fjerning fra tom kø.

(Slutt på oppgave 2)

Oppgave 3: Heap (25%)

En min-heap er et komplett, balansert binært tre der verdiene er ordnet, men ikke sortert. Figur 1 nedenfor viser en min-heap med 12 noder, der verdiene er enkle heltall.



Figur 1

a) Tegn en figur som viser hvordan treet i figur 1 ser ut etter innsetting av verdien 6.

b) Tegn en figur som viser hvordan treet som du tegnet i deloppgave a ovenfor, ser ut etter innsetting av verdien 10.

c) Tegn en figur som viser hvordan treet i figur 1 ser ut etter at den minste verdien er fjernet.

d) En binært tre som er en heap kan lagres effektivt i en array, fordi det ikke er noen «hull» i treet. Her er en array som representerer en heap med 19 noder:

3	5	7	8	10	8	18	17	12	11	10	14	17	22	18	21	15	18	20
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Tegn denne heapen med 19 noder som et binært tre.

(Slutt på oppgave 3)

Oppgave 4: Binært søketre (25%)

I vedlegg 2 er det gitt en Java-klasse `searchTree` . Dette er en uferdig implementasjon av et binært søketre, der verdien i hver node er et enkelt heltall.

I motsetning til i et vanlig binært søketre, skal det *ikke* være tillatt med like verdier i et søketre av klassen `searchTree` . Alle verdiene i treet skal være ulike.

a) I klassen `searchTree` er det gitt en metode for å sette inn en ny verdi i treet:

```
public boolean insert(int value)
```

Denne metoden er ferdig kodet med standardmetoden for innsetting i et binært søketre, som tillater like verdier i treet.

Skriv om metoden `insert` slik at den ikke legger inn en verdi i treet hvis den finnes fra før:

- Hvis metoden kalles med en verdi som ikke finnes fra før, skal denne verdien legges inn på vanlig måte.
- Hvis verdien finnes fra før skal metoden `insert` returnere `false` , og ingen innlegging skal gjøres i treet.

Det holder at du i din besvarelse bare angir den delen av koden der det er endringer.

b) Lag metoden `public int depth(int value)` i klassen `searchTree`. Den skal returnere *dybden* til den noden i treet som inneholder `verdi` . Dette er entydig siden treet ikke har like verdier. Dybden er det samme som avstanden (antall kanter) fra roten ned til noden. Det betyr spesielt at roten har dybde 0. Hvis `verdi` ikke ligger i treet, skal metoden returnere verdien -1.

En *gren* i et binærtre består av alle nodene på en vei fra og med roten og ned til og med en bladnode.

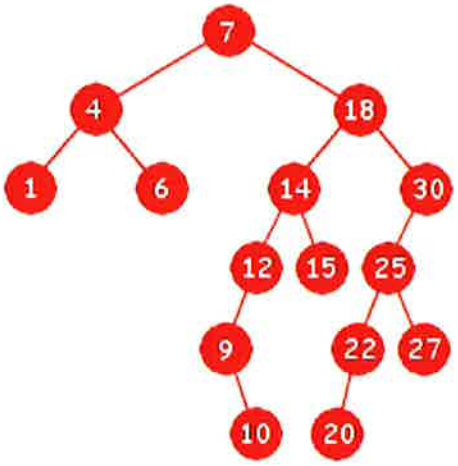
Anta at to forskjellige noder p og q ligger på samme gren i treet. Da sier vi at den av dem som ligger nærmest roten, er en *forgjenger* til den andre. Omvendt sier vi at den av dem som ligger lengst ned, er en *etterkommer* av den andre. Deres *avstand* er antall kanter på veien mellom dem.

Hvis det ikke finnes en gren som inneholder begge de to nodene p og q, så må de to ha en nærmeste felles forgjenger f. I så fall sier vi at avstanden mellom p og q er lik avstanden fra p til f pluss avstanden fra q til f.

Et eksempel som viser avstander i et binært tre er gitt på neste side.

(Oppgave 4 fortsetter på neste side)

Figur 2 viser et binært søketre med 15 noder, der alle 15 heltallsverdier i treet er ulike:



Figur 2

I figur 2 vil avstanden mellom 18-noden (noden med verdi 18) og 10-noden være lik 4. Nærmeste felles forgjenger til 15-noden og 27-noden er 18-noden. Dermed blir avstanden mellom dem (15-noden og 27-noden) lik 2 + 3 = 5.

c) Lag metoden `public int distance(int v1, int v2)` i klassen `searchTree`. Den skal returnere avstanden mellom de to nodene som inneholder verdiene `v1` og `v2` . Hvis en av verdiene (eller begge) ikke ligger i treet, skal metoden returnere -1. Du bestemmer selv hvilken løsningsteknikk du vil bruke, og om du vil bruke hjelpemetoder.

(Slutt på på oppgave 4)

Vedlegg 1: Klassen `twowayQueue`

```
public class twowayQueue                // Toveis kø med enkel
{                                       // lenket liste

   private final static class node      // Indre nodeklasse
   {
      private int value;                // Verdien i noden
      private node next;                // Peker til neste node

      private node(int v, node n)       // Konstruktør
      {
         value = v;
         next = n;
      }
   } // end class node

   private node head;                   // Peker til start på kø
   private int numNodes;                // Antall noder i køen

   public twowayQueue()                 // Konstruktør
   {
      head = null;
      numNodes = 0;
   }

   public void addFirst(int value)      // Sett inn først i kø
   {
      // Skal kodes i oppgave 2 a)
   }

   public int removeFirst()             // Ta ut først i kø
   {
      // Skal kodes i oppgave 2 a)
   }

   public void addLast(int value)       // Sett inn sist i kø
   {
       // Skal kodes i oppgave 2 b)
   }

   public int removeLast()              // Ta ut sist i kø
   {
       // Skal kodes i oppgave 2 b)
   }

} // end class twowayQueue
```

Vedlegg 2: Klassen `searchTree`

```
public class searchTree                    // Søketre med heltall
{

   private static final class node         // En indre nodeklasse
   {
      private int value;                   // Nodens verdi
      private node left, right;            // Venstre og høyre barn

      private node(int v, node l, node r) // Konstruktør
      {
         value = v;
         left = l; right = r;
      }

      private node(int v)                  // Konstruktør
      {
         this(v, null, null);
      }

   } // end class node


   private node root;                      // Roten i treet
   private int n;                          // Antall noder


   public searchTree()                     // Konstruktør
   {
      root = null;
      n = 0;
   }


   public int numNodes()                  // Returnerer antall
   {                                      // verdier i treet
      return n;
   }


   public boolean empty()                 // Sjekker om treet er
   {                                      // tomt
      return n == 0;
   }
```

(Vedlegg 2 fortsetter på neste side)

```
   public boolean insert(int value)       // Setter inn en ny verdi
   {                                      // i treet. Skal skrives
      node p = root, q = null;            // om i oppgave 4 a)
      int cmp = 0;

      while (p != null)
      {
         q = p;
         cmp = value - p.value;
         p = cmp < 0 ? p.left : p.right;
      }
      p = new node(value);

      if (empty())
         root = p;
      else if (cmp < 0)
         q.left = p;
      else
         q.right = p;

      n++;
      return true;
   }


   public int depth(int value)            // Beregner dybden av
   {                                      // en node i treet
      // Skal kodes i oppgave 4 b)
   }


   public int distance(int v1, int v2)    // Beregner avstanden
   {                                      // mellom to noder
      // Skal kodes i oppgave 4 c)
   }


} // end class searchTree
```