Høgskolen i Østfold

EKSAMEN

Emnekode:	**Emnenavn:**
ITF20006	Algoritmer og datastrukturer
Dato:	**Eksamenstid:**
9. mai 2016	9.00 – 13.00
Hjelpemidler:	**Faglærer:**
Alle trykte og skrevne	Jan Høiberg

Om eksamensoppgaven og poengberegning:

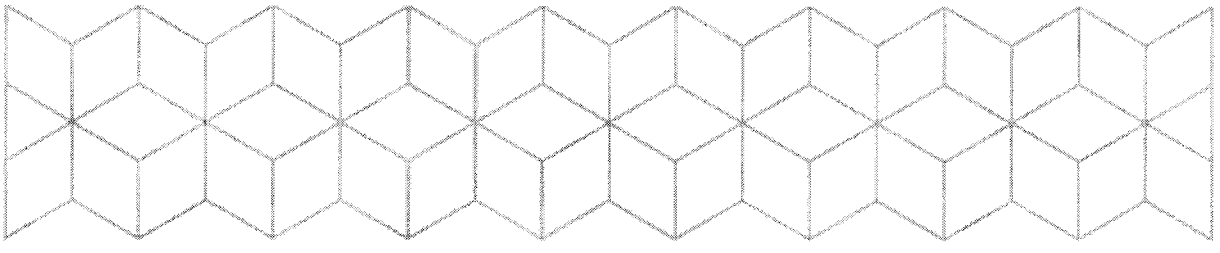
Oppgavesettet består av 7 sider, inkludert denne forsiden. Kontrollér at oppgaven er komplett før du begynner å besvare spørsmålene.

Oppgavesettet består av 4 oppgaver med i alt 16 deloppgaver. Innen hver av de 4 oppgavene vektes alle deloppgaver likt. En av deloppgavene har 5 nummererte underpunkter som også vektes likt. Les hver oppgave nøye før du begynner på besvarelsen.

Programkode i besvarelsen skal skrives i Java. Legg vekt på å skrive ryddig og lett forståelig kode.

Sensurfrist: 30. mai 2016

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



Oppgave 1: Algoritmeanalyse (25%)

For hver metode i deloppgavene a) – f) nedenfor avhenger arbeidsmengden av parameteren n, som er et positivt heltall. Gjør følgende for hver metode:

- Angi metodens arbeidsmengde med *O*-notasjon. Gi en kort begrunnelse for svaret.
- Beskriv kort hva metoden beregner og returnerer.

a)
```
long metode_a(int n)
{
  long f = 1;
  for (int i = 1; i <= n; i++)
     f *= i;
  return f;
}
```

b)
```
long metode_b(int n)
{
   if (n < 2)
      return 1;
   return n * metode_b(n - 1);
}
```

c)
```
int metode_c(int n)
{
   int i = n, l = 0;
   while (i > 0)
   {
      l++;
      i /= 10;
   }
   return l;
}
```

d)
```
long metode_d(int n)
{
  long s = 0;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
       s++;
  return s;
}
```

e)
```
int metode_e(int m, int n)
{
   if (n == 1)
      return m;
   return (m + metode_e(m, n - 1));
}
```

f)
```
long metode_f(int n)
{
   if (n <= 2)
      return 1;
   return metode_f(n - 1) + metode_f(n - 2);
}
```

(Slutt på på oppgave 1)

Oppgave 2: Stack, kø og liste (20%)

a) Forklar kort hvorledes en stack kan implementeres med en enkel-lenket liste slik at operasjonene på stacken er $O(1)$, dvs. at effektiviteten av innsetting og fjerning av data ikke avhenger av hvor mange elementer som er lagret. Tegn enkle figurer som viser hvorledes operasjonene `push` og `pop` implementeres med lenket liste.

b) Forklar kort hvorledes en kø kan implementeres med en enkel-lenket liste slik at operasjonene på køen er $O(1)$. Tegn enkle figurer som viser hvorledes operasjonene `enqueue` og `dequeue` implementeres med lenket liste.

(Slutt på på oppgave 2)

Oppgave 3: Sortering, datastrukturer og effektivitet (25%)

I programmering brukes ofte betegnelsen "black box" om programvare der bare grensesnittet/ funksjonaliteten er kjent, mens selve "innmaten"/implementasjonen ikke er tilgjengelig. I denne oppgaven skal du skrive et lite program som bruker en slik "black box".

Følgende Java-klasse, som implementerer en datastruktur for å lagre heltall, er gitt:

```
public class blackBox
{
    ...

    public blackBox(int n)
    {
        // Konstruktør som oppretter en black box som kan
        // lagre opp til n heltall
        ...
    }

    public void insert(int value)
    {
        // Metode for å sette inn et nytt heltall i en
        // black box
        ...
    }

    public int removeMin()
    {
        // Metode for å fjerne minste heltall fra en black box
        // Returnerer verdien som fjernes
        ...
    }
}
```

Symbolet `...` brukes ovenfor til å angi at selve koden som implementerer klassen og metodene ikke er tilgjengelig.

I programmet du skal skrive nedenfor, kan du anta at du har full tilgang til å kunne bruke metodene i klassen `blackBox`.

a) Programmér følgende metode:

```
void blackBoxSort(int a[])
```

Metoden `blackBoxSort` skal sortere arrayen `a`, ved å bruke de tre metodene som er angitt for klassen `blackBox` ovenfor. Metodene `insert` og `removeMin` skal begge kalles nøyaktig like mange ganger som det er elementer i arrayen `a`.

(Opppgave 3 fortsetter på neste side)

Effektiviteten til metoden `blackBoxSort` vil avhenge av hvor effektivt klassen `blackBox` implementerer innsetting og fjerning av verdier i datastrukturen.

b) Anta at arrayen som skal sorteres inneholder n elementer. Angi arbeidsmengden for metoden `blackBoxSort`, uttrykt med O-notasjon, når klassen `blackBox` er implementert som:

1. Usortert array
2. Usortert lenket liste
3. Vanlig binært søketre (ikke selvbalanserende)
4. Heap
5. B-tre av orden 4

For hvert av tilfellene 1 – 5 ovenfor skal det gis en kort begrunnelse for svaret.

c) Kan klassen `blackBox` implementeres effektivt som en hashtabell? Begrunn svaret.

(Slutt på opppgave 3)

Oppgave 4: Binære søketrær (30%)

I denne oppgaven brukes det et binært søketre som lagrer data om personer. For hver person skal det lagres:

- Etternavn (en tekststreng)
- Fornavn (en tekststreng)
- Alder (et heltall)

Søketreet er alfabetisk sortert på etternavn. For personer med samme etternavn, ligger fornavn og alder for hver person lagret i en sortert, lenket liste, sortert alfabetisk på fornavn. Det er én slik liste for hver node i treet.

En tegning som viser et eksempel på denne datastrukturen er gitt i figur 1 på neste side.

Nodene i søketreet skal være av klassen `treNode`. Nodene i lenkede lister skal være av klassen `listeNode`.

a) Skriv Java-kode for de to klassene `treNode` og `listeNode`, slik at de bare inneholder variablene som trengs for å lagre datastrukturen beskrevet ovenfor.

b) Skriv en konstruktør for klassen `listeNode` som setter fornavn og alder til gitte verdier. Andre variable i klassen skal også få fornuftig(e) verdi(er).

c) Skriv en konstruktør for klassen `treNode` som har som parameter fornavn, etternavn og alder til en person. Konstruktøren skal bruke metoden du skrev i forrige deloppgave.

d) Skriv en rekursiv metode:

```
void skrivPersoner(treNode rot)
```

som skriver ut en linje med data for hver person som er lagret i treet med rot i noden `rot`. En linje med utskrift kan se slik ut:

```
Høiberg, Jan (42)
```

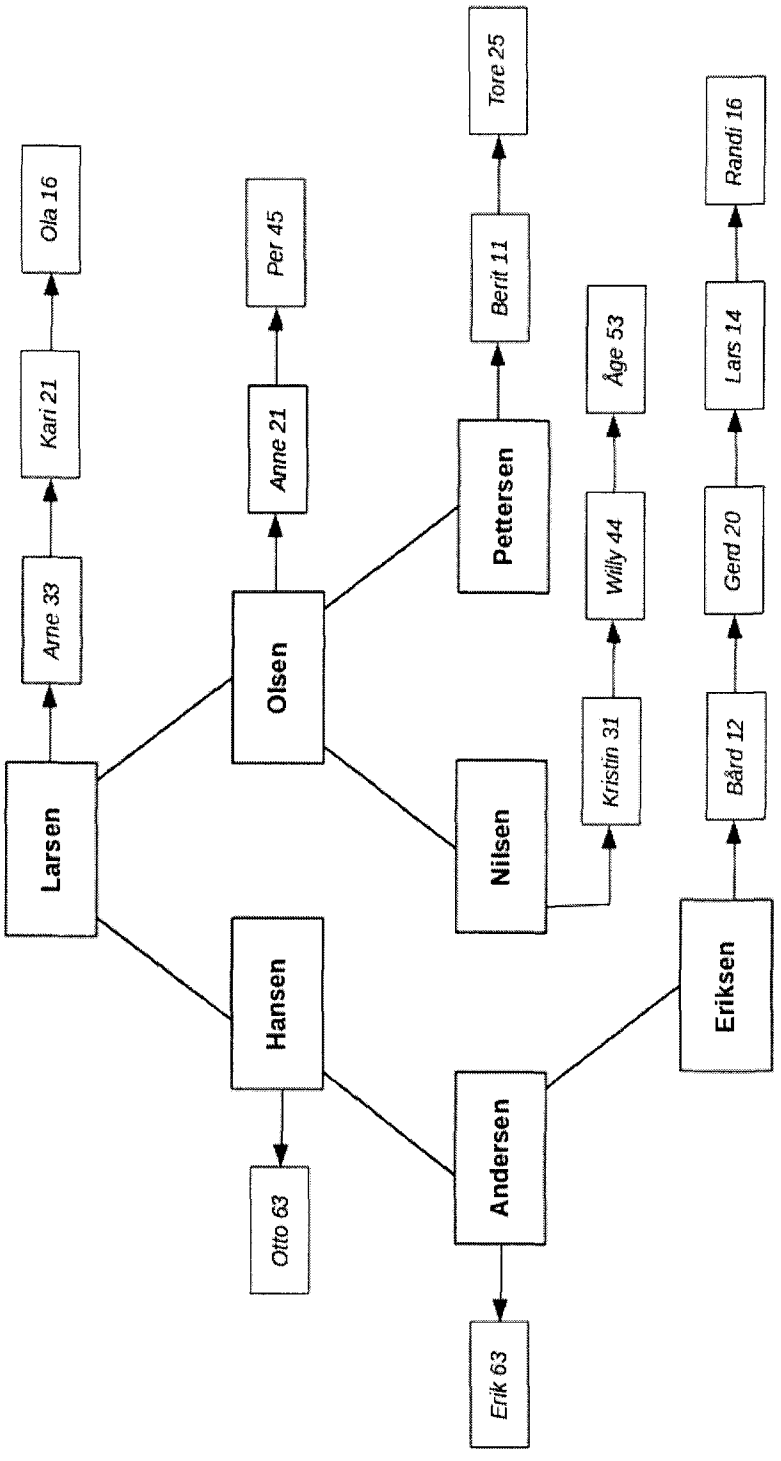
Utskriften skal være sortert på etternavn. Personer med samme etternavn skal skrives ut sortert på fornavn.

e) Skriv en rekursiv metode:

```
boolean finnes(treNode rot, String fornavn, String etternavn)
```

som returnerer `true` hvis det finnes en person lagret i treet med rot i noden `rot`, som har fornavn lik verdien av parameteren `fornavn` og etternavn lik verdien av parameteren `etternavn`. Hvis denne personen ikke finnes, skal metoden returnere `false`. Metoden `finnes` skal være så effektiv som mulig.

(Slutt på opppgave 4)


Larsen
Hansen
Olsen
Nilsen
Pettersen
Andersen
Eriksen
Ola 16
Kari 21
Arne 33
Per 45
Anne 21
Tore 25
Berit 11
Åge 53
Willy 44
Kristin 31
Randi 16
Lars 14
Gerd 20
Bård 12
Otto 63
Erik 63

Figur 1