Høgskolen i Østfold

# EKSAMEN

| **Emnekode:**<br><br>ITF20006 | **Emne:**<br><br>Algoritmer og datastrukturer |
|---|---|
| **Dato:**<br>22. mai 2007 | **Eksamenstid:**<br>kl 09.00 til kl 13.00 |
| **Hjelpemidler:**<br>4 A4-ark med valgfritt innhold på begge sider. | **Faglærer:**<br>Mari-Ann Akerjord |

**Eksamensoppgaven:**
Oppgavesettet består av 10 sider inklusiv vedlegg og denne forsiden.

Oppgavesettet består av 3 oppgaver, med henholdsvis 3, 5 og 4 deloppgaver. Alle oppgavene skal besvares. Ved sensur teller hver deloppgave omtrent like mye. Karakter fastsettes dog på basis av en helhetsvurdering av besvarelsen.

**Kontroller at oppgaven er komplett før du begynner å besvare spørsmålene!**

Sensurdato: 13. juni 2007

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

# Oppgave 1

**A.** Legg verdiene 30 47 2 81 61 35 1 11 16 inn i et på forhånd tomt binært søketre. Tegn treet du ender opp med og skriv deretter ut elementene i postorden rekkefølge.

**B.** I Vedlegg 1 finner du et enkelt HTML-dokument. Et slikt dokument består av tagger og tekst. Et krav til at HTML- og XML-dokumenter skal være såkalt velformede, er at det for hver start-tagg finnes en slutt-tagg. I tillegg må taggene være korrekt nøstet. Dokumentet i Vedlegg 1 oppfyller disse kravene.

Anta at du har tilgjengelig en HTML-leser, som leser ett og ett element fra en HTML-fil. Elementene som returneres fra leseren kan være av 3 forskjellige typer:

1. start-tagg, for eksempel <h1>
2. slutt-tagg, for eksempel </h1>
3. tekst , for eksempel ”Dette er en overskrift”

Andre typer elementer neglisjeres av leseren. Hvis elementet er en start- eller slutt-tagg, har du dessuten tilgang til taggens navn. Navnene til en start-tagg og en slutt-tagg som hører sammen er identiske. For eksempel er ”h1” navnet til taggene <h1> og </h1>.

Anta videre at du har en implementasjon av en stakk. Interfacet `StackADT`, som beskriver de vanlige operasjonene på en stakk, er vist i Vedlegg 1. Beskriv med pseudokode en algoritme som benytter HTML-leseren og en stakk til å avgjøre om et HTML-dokument er velformet eller ikke.

**C.** Nedenfor ser du et lite program bestående av metodene `algoritme1` og `algoritme2`, samt en `main`-metode.

```
public class Algoritmer{

  public static void algoritme1(int n){
     int a=0;
     for (int i=0; i< n; i++)
        for (int j=i; j < n; j++)
           a++;
     System.out.println(a);
  }

  public static void algoritme2(int n){
     if (n==0)
        return;
     algoritme2(n-1);
     System.out.print(n+" ");
  }

  public static void main(String [] args){
     algoritme1(10);
     System.out.println("\n");
     algoritme2(10);
  }
}
```

1. Hva skrives til skjerm når programmet kjøres?

2. Angi arbeidsmengde i O-notasjon for metodene algoritme1 og algoritme2. Begrunn svarene.
3. Anta at du har målt kjøretiden T(N) for hver av de to metodene for en bestemt verdi av oppgavenes størrelse N. Maskinen din blir deretter oppgradert med en CPU som utfører hver operasjon dobbelt så raskt. Vil du nå kunne doble oppgavenes størrelse og forvente å bli ferdig på samme tid som før? Begrunn svarene.

# Oppgave 2

I Vedlegg 2 finner du klassen `SortertListe`, som skal kunne brukes til å lage sorterte lister av hvilken som helst objekttype som implementerer interfacet `Comparable`. Som intern datastruktur benyttes en enkeltlenket liste. Elementene i listen er av type `Node`, som er en privat, indre klasse i klassen `SortertListe`. Listen skal være sortert i stigende rekkefølge, det vil si med elementet med lavest verdi først. Videre kan listen inneholde identiske verdier, det vil si at duplikater tillates.

**A.** Metoden `fjernMinste` skal fjerne elementet med lavest verdi og returnere verdien til dette elementet. Implementer metoden `fjernMinste`.

**B.** Metoden `skriv` skal skrive til skjerm verdiene i den sorterte listen i stigende rekkefølge. Implementer metoden `skriv`.

**C.** Metoden `settInn` skal sette inn en ny verdi på riktig plass i listen. Verdien som skal settes inn er parameter til metoden. Implementer metoden `settInn`.

**D.** Metoden `flett` skal flette sammen denne sorterte listen med listen gitt som parameter. Resultatet skal returneres som et *nytt* `SortertListe`-objekt, det vil si at ingen av de to listene som flettes sammen skal endres som følge av denne operasjonen. Implementer metoden `flett`. Hvor effektiv er algoritmen din? Kan den gjøres mer effektiv?

**E.** I Vedlegg 2 finner du også interfacet `PrioritetsKø`. Lag en klasse `ListePrioritetsKø` som implementerer dette interfacet ved å benytte en `SortertListe` som intern struktur.

# Oppgave 3

**A.** Et binært tre er en minimumsheap hvis treet er komplett, og hvis vi for hver node har at nodens verdi er mindre enn verdiene i nodens eventuelle barn. Denne definisjonen tillater ikke duplikater.

Hvis vi nummererer nodene i et slikt tre i nivåordnet rekkefølge (levelorder), kan vi for enhver node beregne nummeret til foreldrenoden og begge barna. Vi antar at nummeret på rota er 1, og at node nummer *k* ikke er rota i treet og at den har to barn.

- Hva er nummeret til foreldrenoden til node *k*?

- Hva er nummeret til venstre barn av node *k*?
- Hva er nummeret til høyre barn av node *k*?

**B.** Nedenfor ser du en minimumsheap representert i en tabell:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 21 | 16 | 24 | 30 | 19 | 68 | 65 | 26 | 41 | | | | |

Tegn det korresponderende binærtreet.

Når vi skal ta ut en node eller sette inn en ny node i en minimumsheap, må vi passe på at egenskapene til heapen bevares. Anta at du har en fungerende heap-implementasjon, med metoder for å fjerne minste element fra heapen og å sette inn et nytt element. Vis hvordan treet ser ut etter at følgende operasjoner er utført, når *heap* er heapen representert ved tabellen over:

```
heap.settInn(29);
heap.settInn(15);
heap.fjernMinste();
```

På grunn av egenskapene beskrevet over, implementeres ofte minimumsheaper ved bruk av tabeller. Men det er selvsagt også mulig å implementere et slikt tre med en tradisjonell referansestruktur med noder, venstre og høyre barn. I Vedlegg 3 finner du klassen `MiniHeap` som skal representere en minimumsheap implementert med en referansestruktur. Nodene i treet er av typen `Node`, som er en privat, intern klasse i klassen `MiniHeap`.

Den ene konstruktøren i klassen skal gjøre det mulig å konstruere treet ut fra en heap gitt på tabellform (jfr. forrige oppgave). Tabellen som skal konverteres til referansestruktur, er parameter til konstruktøren. Første element er tomt, og rota til heapen ligger i posisjonen med indeks 1.

**C.** Før treet konstrueres, kalles to metoder for å avgjøre om tabellen som mottas av konstruktørens parameter representerer en minimumsheap. I motsatt fall kastes et unntak, og treet bygges ikke. De to metodene er:

1. `erKomplett` – som skal avgjøre om tabellen representerer et komplett binært tre. Det vil si at tabellen må være fyllt opp fra venstre mot høyre. Med unntak av første element som er tomt, kan ikke tabellen ha noen tomme plasser (null-referanser) som etterfølges av en plass som inneholder en verdi. Tabellen trenger imidlertid ikke å være fyllt helt opp, men kan for eksempel se ut som tabellen vist i oppgaven over.

2. `erMiniHeap` – som skal avgjøre om elementene i tabellen er ordnet i henhold til regelen som gjelder for en minimumsheap.

Implementer metodene `erKomplett` og `erMiniHeap`. Velg selv om du vil implementere sistnevnte metode iterativt eller rekursivt.

**D.** Metoden `lagMiniHeap` skal konvertere tabellen gitt som parameter til en heap implementert med noder som har referanser til høyre og venstre barn. Metoden skal fungere som en driverrutine for den rekursive metoden `lagSubtre`. Bestem selv hvilke parametere denne metoden skal ha.

# Vedlegg 1

```
-----------------------------------------------------------------------------------------------
<html>
   <head>
      <title>
         En tittel.
      </title>
   </head>
   <body>
      <h1> Dette er en overskrift</h1>
      <p>
         Et avsnitt.
      </p>
      <p>
         Et avsnitt til.
      </p>
   </body>
</html>
-----------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------
public interface StackADT<T> {

   // Adds one element to the top of this stack
   public void push(T element);

   // Removes and returns the top element from this stack
   public T pop();

   // Returns without removing the top element of this stack
   public T peek();

   // Returns true if this stack contains no elements
   public boolean isEmpty();

   // Returns the number of elements in this stack
   public int size();
}
-----------------------------------------------------------------------------------------------
```

# Vedlegg 2

```
public class SortertListe<T extends Comparable<? super T>> {

   private Node<T> start;
   private int antall;


   public int hentAntall(){return antall; }


   /**
    * Fjerner og returnerer minste verdi i listen
    * @return minste verdi i listen
    */
      public T fjernMinste(){
     // Oppgave 2A
   }


   /**
    * Skriver listen i stigende rekkefølge
    */
   public void skriv() {
      // Oppgave 2B
   }


    /**
    * Setter inn en ny verdi på riktig plass i den lenkede listen
    * @param verdi verdien som skal settes inn
    */
   public void settInn(T verdi){
               // Oppgave 2C
   }



    /**
    * Fletter denne listen sammen med listen gitt som parameter
        * og returnerer resultatet som en ny SortertListe
    * @return ny Ny SortertListe
        */
   public SortertListe<T> flett(SortertListe<T> l){
      // Oppgave 2D
   }


   /**
    * Klassen Node representerer en node i den lenkede listen
    */
   private class Node<T>{
```

```
        private T verdi;
        private Node<T> neste;

        public Node(T verdi, Node<T> neste){
            this.verdi = verdi;
            this.neste = neste;
        }

    }// slutt class Node

}//slutt class SortertListe
```

```
public interface PrioritetsKø<T> {

   // Legger inn en ny verdi i prioritetskøen
   public void settInn(T verdi);

   // Tar ut verdien med høyest prioritet (lavest verdi)
   public T taUt();

   // Antall elementer i køen
   public int antall();

}
```

# Vedlegg 3

```
/*
 * MiniHeap.java
 */


public class MiniHeap<T extends Comparable<? super T>> {

   private Node<T>  rot;
   private int antall = 0;


   public MiniHeap() {
      rot = null;
      antall=0;
   }


   /** Konstruktør som konverterer heap fra tabellform til referanseform
    * @param heap heap på tabellform
    */
   public MiniHeap(T [] heap)throws Exception {


      // Avgjør om tabellen representerer et komplett binærtre
      if (!erKomplett(heap))
         throw new Exception("Feil! Ikke et komplett tre. ");

      // Avgjør om tabellen representerer en minimumsheap
      if (!erMiniHeap(heap))
         throw new Exception("Feil! Ikke et minimumstre. ");
```

```
        // Konverterer heap fra tabell-representasjon til
        // referanse-representasjon:
        lagMiniHeap(heap);
    }


    /**
     * Metode som avgjør om heapen er komplett
     * @param heap heap på tabellform
     */
  private boolean erKomplett(T[] heap){
    // Oppgave 3C
  }


  /**
   * Metode som avgjør om heapen er en minimumsheap.
   * Driverrutine for rekursiv metode med samme navn.
   * @param heap heap på tabellform
   */
  private boolean erMiniHeap(T [] heap){
    // Oppgave 3C
  }


  /**
    * Metode som konverterer heap på tabellform til heap på referanseform
    * Driverrutine for den rekursive metoden lagSubtre
    * @param heap heap på tabellform
    */
  private void lagMiniHeap(T [] heap){
    // Oppgave 3D
  }



  private Node<T> lagSubtre(<Bestem parametere selv!>){
    // Oppgave 3D
  }



    /** Klassen Node representerer en node i heapen. */
  private class Node<T> {

    private T element;
    private Node<T> venstre;
    private Node<T> hoyre;

    public Node(T element, Node<T> venstre, Node<T> hoyre){
      this.element = element;
      this.venstre = venstre;
      this.hoyre = hoyre;
    }
  }// Slutt, class Node

} //Slutt, class MiniHeap
```