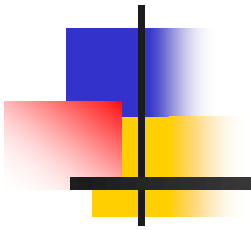


Prøveeksamen INF2440 v 2016

Arne Maus
PSE,
Inst. for informatikk

Oppgave 1 (10 poeng)

Forklar hva som skjer ved en synkronisering:
a) Når to tråder synkroniserer på samme synkroniseringsobjekt (f.eks en ReentrentLock).
b) Når to tråder synkroniserer på hvert sitt synkroniseringsobjekt (f.eks to ReentrentLock).

a) En av trådene må vente hvis de to trådene samtidig prøver å synkronisere og den første tråden ikke er 'ferdig' – har sagt ls1.lock(); men enda ikke har sagt ls1.unlock();
b) Ingenting. De to trådene kan si lock() og unlock() men det medfører ingen venting fra den andre tråden.

Oppgave 2 (25 poeng)

Anta at du har tre tråder som prøver å oppdatere to globale variable x1 og x2, deklarert i A-området Modell2-koden (vedlagt), begge initielt =0 i, slik:

```
public void run( ) {
    for (int i =0; i < 2; i++){
                x1++;
                x2++;
                System.out.println("  - tråd:"+ind+", x1:"+x1+", x2:"+x2 );
    }
     System.out.println(" Tråd:"+ind+" terminerer"+", x1:"+x1+", x2:"+x2);
  } // end run:
```

Like etter at man i metoden utfoer(..) har sagt join() på de tre trådene er følgende utskrifttssetning:

```
System.out.println("Main-tråden  TERMINERER"+", x1:"+x1+", x2:"+x2 );
```

Hver gang man kjører dette programmet, får man nesten alltid veldig forskjellige utskrifter. Her er to:

```
U1:
- tråd:2, x1:2, x2:3
- tråd:1, x1:2, x2:3
- tråd:1, x1:4, x2:5
- tråd:0, x1:2, x2:3
Tråd:1 terminerer, x1:4, x2:5
- tråd:2, x1:3, x2:4
Tråd:2 terminerer, x1:5, x2:6
- tråd:0, x1:5, x2:6
Tråd:0 terminerer, x1:5, x2:6
Main-tråden  TERMINERER, x1:6, x2:6
```

```
U2:
- tråd:1, x1:2, x2:2
  - tråd:2, x1:3, x2:3
  - tråd:2, x1:5, x2:5
  - tråd:0, x1:1, x2:1
  Tråd:2 terminerer, x1:5, x2:5
  - tråd:1, x1:4, x2:4
  Tråd:1 terminerer, x1:6, x2:6
  - tråd:0, x1:6, x2:6
  Tråd:0 terminerer, x1:6, x2:6
Main-tråden  TERMINERER, x1:6, x2:6
```

Oppgave:

Forklar og begrunn kort svarene på følgende spørsmål:

a) Hvorfor ser f.eks tråd:2 i første linje ulike verdier og x1 < x2 i U1?
b) Hvorfor ser ingen av trådene x1 ==6, mens main-tråden ser x1== 6 til sist i U1.
c) Hvorfor ser ingen av trådene i U1 verdien 1?
d) Vil alltid main-tråden skrive ut x1 ogx2 med verdien 6?
e) Hvorfor er utskriftene ulike (hvorfor 'aldri samme resultat' på gjentatte kjøringer)?

Oppgave:

Forklar og begrunn kort svarene på følgende spørsmål:

a) Hvorfor ser f.eks tråd:2 i første linje ulike verdier og x1 < x2 i U1?

Tråd:1 har startet og øket både x1 og x2 til 2, MEN verdien av x1==2 har ikke kommet fram til Tråd:2 (bare x1 == 1) – derfor x1==2, x2 == 3.

Husk at System.out.println(..) er ikke en synchronized metode.

b) Hvorfor ser ingen av trådene x1 ==6, mens main-tråden ser x1== 6 til sist i U1.

join() på trådene er en synkronisering, derfor ser main-tråden alle de 'siste' verdiene.

c) Hvorfor ser ingen av trådene i U1 verdien 1?

Fordi minst to av Tråd:0-2 har startet **før** første utskriftssetning og de har økt både x1 og x2 med 1.

d) Vil alltid main-tråden skrive ut x1 ogx2 med verdien 6?

Nei – vi kunne fått x++ feilen her , dvs. at en av oppdateringene av x1,x2 går ekte tapt.

Vi må skille mellom at trådene ser 'gamle' verdier og at oppdateringer går tapt selv etter synkronisering.

e) Hvorfor er utskriftene ulike (hvorfor 'aldri samme resultat' på gjentatte kjøringer)?

Fordi System.out.println tar lang tid og ofte vil gi sikkert at disse trådene går samtidig, men med uforutsigbare tidsforsinkelser.

Oppgave 3 (25 poeng)

Lag et parallelt program med to tråder og to **CyclicBarrier** hvor de to trådene vekselsvis sender en nummerert melding til den andre tråden 20 ganger (f.eks. «Tråd0 sier: Hei nr 14») . Meldingen skrives ut med **System.out.println(...)**. Når den ene tråden skriver, skal den andre tråden vente på en av de to **CyclicBarrier-**ene og motsatt.

Bare skriv run() –metoden og initieringen av de to **CyclicBarrier-**ene.

```
public void run( ) {
    for (int i =0; i < 20; i++){
            try { b.await(); } catch (Exception e){return;}
              if (ind==0)   System.out.println("  - tråd:"+ind+" sier Hei nr:"+i );
             try { c.await();  } catch (Exception e){return;}
              if (ind==1)   System.out.println("  - tråd:"+ind+" sier Hei nr:"+i );
     } // end hei-loop

 } // end run:
```

Kunne denne oppgaven vært løst med bare én CyclicBarrier ?

Oppgave 4 (15 poeng)

Anta at du har to tråder som aksesserer en felles variabel, deklarert i felt A slik: **int i =0,** og at hver av trådene har deklartert i det felles området A for alle trådene:

ReentrantLock lock = new ReentrantLock(2);

Tråd0 utfører følgende kode:

```
lock.lock();
 try {   i++;
} finally { lock.unlock();}
i++;
```

før den terminerer, mens

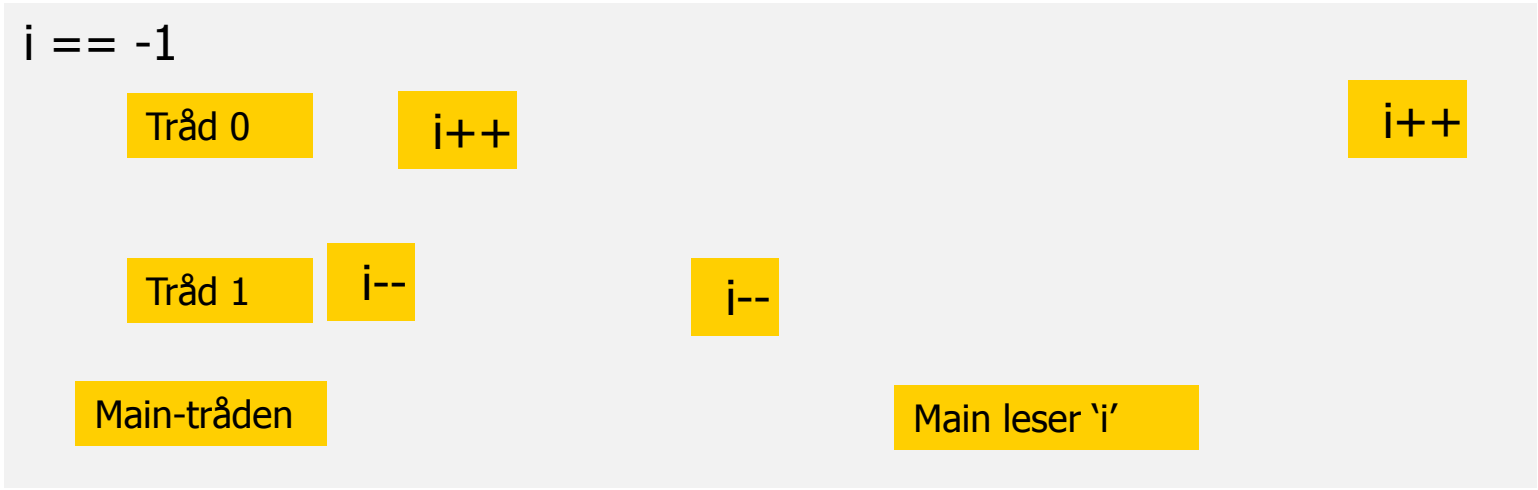
Tråd1 utfører følgende kode:

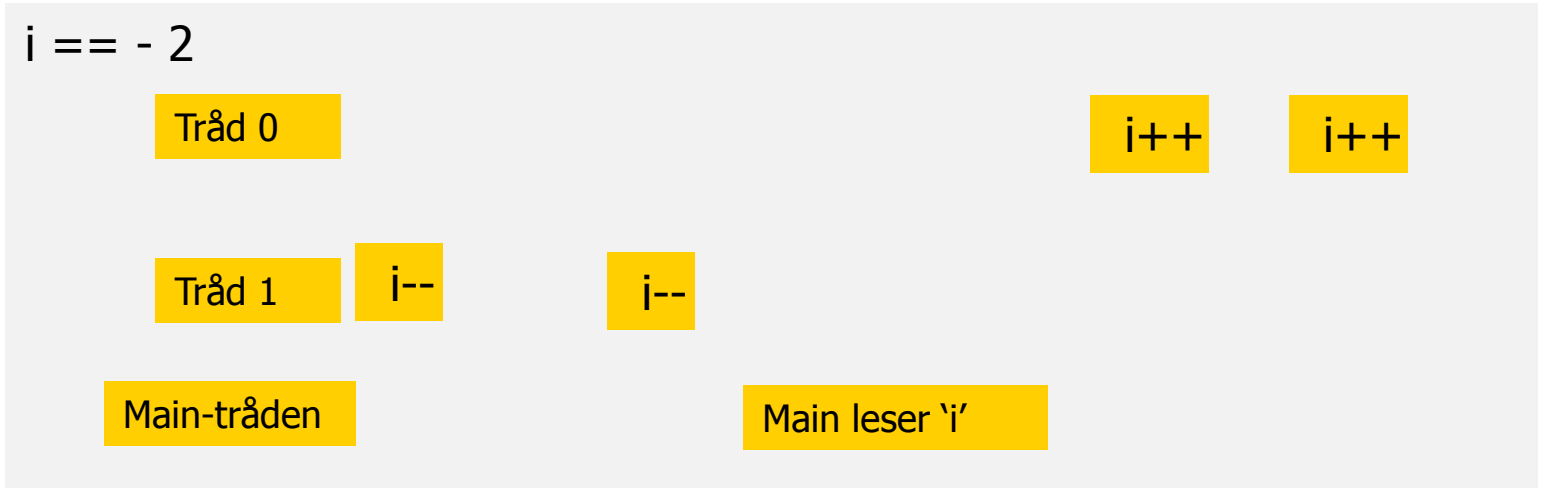
```
i--;
lock.lock();
      try {  i--;
} finally { lock.unlock();}
```

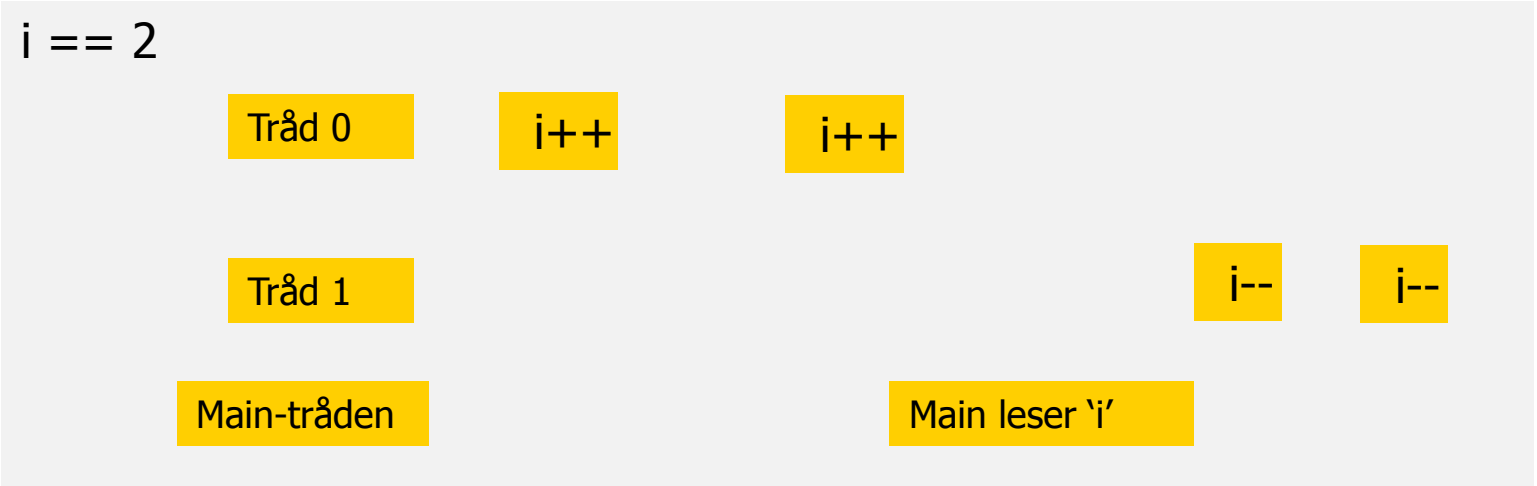
før den terminerer,

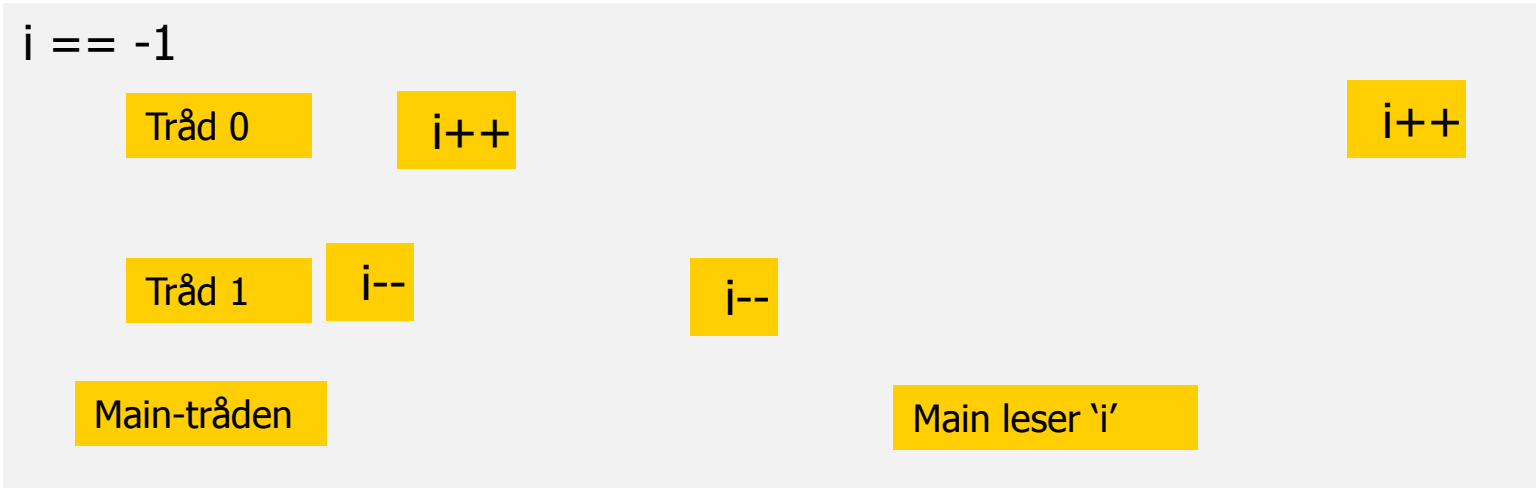
Hvilke mulige verdier kan **i** ha etter at begge trådene har terminert?.

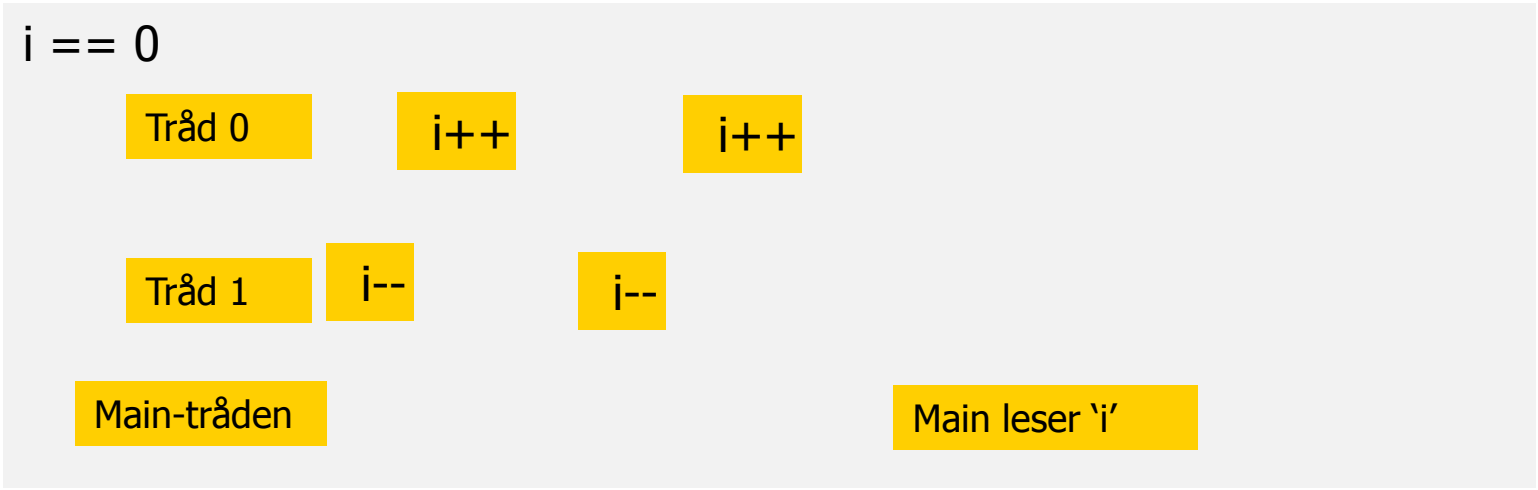
Mulige verdier av i : -2,-1,0,1,2


i == -1
Tråd 0
i++
i++
Tråd 1
i--
i--
Main-tråden
Main leser 'i'


i == - 2
Tråd 0
i++
i++
Tråd 1
i--
i--
Main-tråden
Main leser 'i'


i == 2
Tråd 0
i++
i++
Tråd 1
i--
i--
Main-tråden
Main leser 'i'


i == -1
Tråd 0
i++
i++
Tråd 1
i--
i--
Main-tråden
Main leser 'i'


i == 0
Tråd 0
i++
i++
Tråd 1
i--
i--
Main-tråden
Main leser 'i'

Oppgave5 (30 poeng)

Anta at du har deklarert en array: **int tallene[] = new int [n];** og du kan anta at elementene i **tallene[]**er sortert ***stigende***.

Du skal nå skrive først en sekvensiell og så en parallell metode (anta at du har k kjerner) som snur innholdet av **tallene[]**slik at tallene blir sortert ***synkende***. Skriv heller ikke her hele programmet, men bare de metodene og evt. data som du trenger i A og B området i vedlegget + hva innholdet av run()-metoden i trådene er.

```
void snu ( int [] a, int left, int right) {
    int an = a.length-1, t;

    for (int i = left; i < right; i++) {
              t = a[i];
              a[i] = a[an-i];
              a[an-i] =t;
    }
}// end snu
```

```
Arbeider (int in, int [] a) {
      ind = in;
      this.a =a;
       num = a.length/(2*antT); // only swap half
       left = ind*num;
       right = (ind+1)*num;
        if (ind == antT -1) right += (a.length%antT)/2;
  }

    public void run( ) {
          // kalles når tråden er startet
         snu(a, left, right);
     } // end run
```

Oppgave 6 (40 poeng)

Du har gitt en metode for å sortere (i Appendix II er en slik algoritme gitt basert på innstikk-sortering, men vi kunne like godt ha brukt radix eller quicksort i steden):

public static void sort (int a[], int left, int right) {..}

Du skal nå skrive en ny metode **sort2**, basert på denne som sorterer to arrayer med følgende forståelse:

public static void sort2 (int a[], int [] b, int left, int right) {..}

Du sorterer innholdet av arrayen **a[]** som i den første metoden "**sort**" , men med det tillegget at elementene i den andre arrayen **b[]** flyttes helt tilsvarende som elementene i **a[]** flyttes.

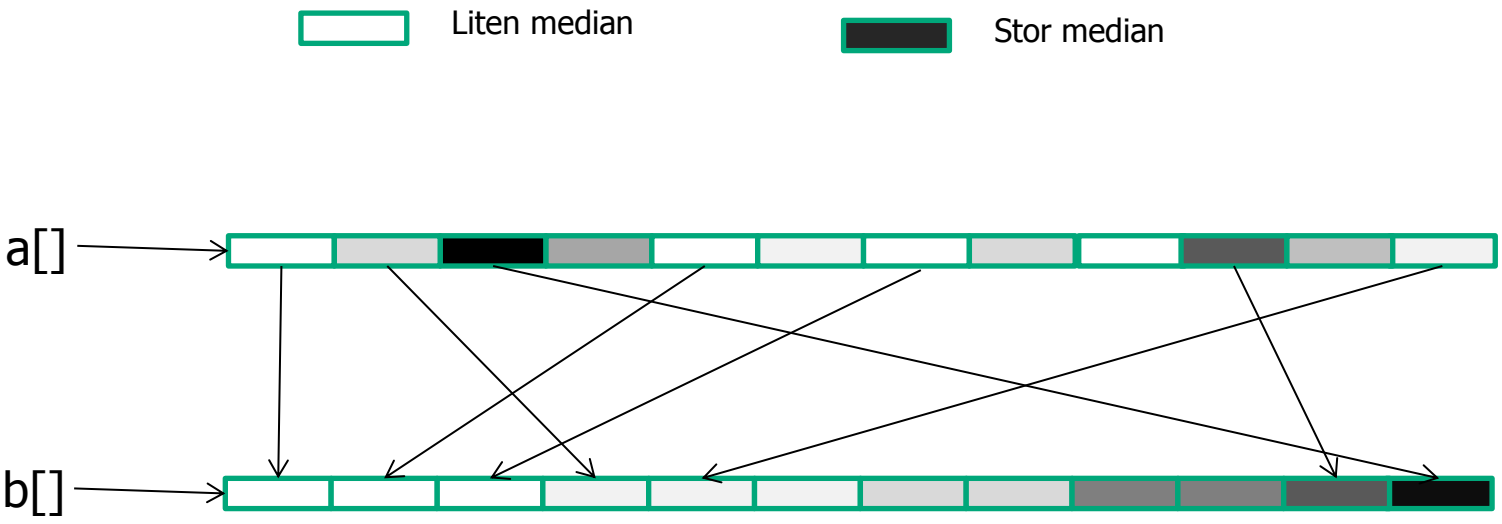
Eksempel: hvis elementet **a[0]** ble flyttet til **a[23]** etter sortering så vil også det opprinnelige elementet **b[0]** nå stå på plass **b[23]**. Dette oppnår du ved at hver gang du flytter **a[]** – elementene, flytter du **b[]** elementene med samme indeks helt tilsvarende.

Skriv en sekvensiell metode "**sort2**" som løser denne oppgaven. Denne vil du så bruke for å løse oppgave 7.

```
/** sorts a [left .. right]  and b [left.. right] by Insertionsort.*/
 void sort2 (int a[],int []b, int left, int right) {
            int i,k;
            int t,tb;

            for (k = left ; k < right; k++) {
                        if (a[k] > a[k+1]){
                                    t =   a[k+1];
                                    tb = b[k+1];
                                    i = k;

                                     while (i >= left && a[i] >t {
                                                a[i+1] = a[i];
                                                b[i+1] = b[i];
                                                i--;
                                    }

                                     a[i+1] = t;
                                     b[i+1] = tb;
                         } // end if
             }// end k
 } // end Sort2
```

Oppgave 7 (80 poeng)

Du har en foreleser som tror at han har en genial idé til å lage en raskere innstikksortering-metode kalt **medianSort**. Problemet med innstikksortering, tenkte han, er at det er en del store elementer tidlig (med lave indekser) i arrayen som må skyves *langt* avgårde mot enden av arrayen, og motsatt, mange små verdier alt for langt ut i array-en som sorteres mot begynnelsen. Disse må flyttes (byttes med hverandre) før vi gjør instikksortering til sist, som da vil gå svært mye raskere.



Den sekvensielle algoritmen for dette er slik:

a) Vi starter med å dele opp arrayen i en rekke m mindre områder (f.eks i m = $n/5$ deler) og du kan for enkelthets syld anta at n er delbar med m uten rest (n%m ==0).
   Vi sorterer så hver og en av dem med "**sort**" fra Appendix II samtidig som vi noterer verdien av det midterste elementet i hver av de sorterte delene i: int[] median = new int[m]. Samtidig har du laget en annen array
   int[] index = new int [m] som initielt inneholder 0,1,2,..,m-1.
b) Sorter nå arrayene median og index med **sort2** fra oppgave 6. Du kan her anta at du har løst oppgave 6 om å lage en sortering for to arrayer slik at når du sorterer på den ene arrayen (her: median) og har med en annen array indeks (initielt: 01,2,…,$m-1$) som parameter, så deltar elementene i indeks i de samme ombyttingene som elementene i median. Når da median-arrayen er sortert, vil f.eks indeks[0] si hvilken av delene av a[] som hadde minst median, indeks[1] hvilken som hadde nest minst median, … osv.
c) Vi flytter om små-områdene i a[] over i en like lang array b[] slik at det området som har minst median kommer først i b[], nest minst kommer som område nr. 2, osv
d) Når vi er ferdige med alle disse omflyttingene kopieres b[] tilbake til a[] og
e) det gjøres ett kall på sort fra AppendixII av hele arrayen a[].

Den sekvensielle algoritmen for dette er slik:

a) Vi starter med å dele opp arrayen i en rekke m mindre områder (f.eks i m = $n/5$ deler) og du kan for enkelthets syld anta at n er delbar med m uten rest (n%m ==0).
Vi sorterer så hver og en av dem med "**sort**" fra Appendix II samtidig som vi noterer verdien av det midterste elementet i hver av de sorterte delene i: int[] median = new int[m]. Samtidig har du laget en annen array int[] index = new int [m] som initielt inneholder 0,1,2,..,m-1.

```
// median-for small sections
int medianVal (int [] a, int index)  {
      int start = index*MED_LEN, end = start +MED_LEN-1;
      insertSort(a,start ,end);
     return a[(start+ end)/2];
} // end int medianVal
```

Den sekvensielle algoritmen for dette er slik:

c) Vi flytter om små-områdene i a[] over i en like lang array b[] slik at det området som har minst median kommer først i b[], nest minst kommer som område nr. 2, osv

```
void copySegments(int[] a,int[] b, int index[], int i){
        int  aStart = i*MED_LEN,
         bStart = index[i]*MED_LEN;

        for (int j = 0; j < MED_LEN; j++) {
                b[bStart+j]= a[aStart+j];
         }// end j
 } // end copy segmnent
```

Den sekvensielle algoritmen for dette er slik:

d) Når vi er ferdige med alle disse omflyttingene kopieres b[] tilbake til a[] og

```
void copyBack (int [] a, int [] b,int startSeg, int endSeg) {
    int start = startSeg*MED_LEN, end = endSeg*MED_LEN;

    for (int j = start; j < end; j++) {
                a[j]= b[j];
    } // end j
} // end copyback
```

```
// sequential  median sort of a[]
void medianSort (int [] a) {
        int antMed = (a.length)/MED_LEN, start,end;

       // steg  a)
        for (int i = 0 ; i < antMed; i++){
              median[i] = medianVal(a,i);
        }
        // steg b)
        sort2 (median,index,0,antMed-1); // sort median , move index same

        // steg c), copy the differnt parts to their median positions in b
        for (int i = 0 ; i < antMed; i++){
              copySegments (a,b,index,i);
         }

          // steg d)
            copyBack (a,b,0,antMed);

          // steg e)
          insertSort (a);
  } // end medianSort
```

```
public void run( ) {
    int antMed = (a.length)/MED_LEN;
    num = antMed/antT;
    start = ind*num;
    end = (ind+1)*num;
    if (ind == antT -1) end += antMed % antT;

     // steg  a)
    for (int i = start ; i < end; i++){
                median[i] = medianVal(a,i);
    }
    // ---------- sync 1 ------------------------------
    try { barrier.await();} catch(Exception e) {return;}

    // steg b)
     if (ind ==0) sort2 (median,index,0,antMed-1); // sort median,index

     // ---------- sync 2 ------------------------------
     try { barrier.await();} catch(Exception e) {return;}

     // steg c),
      for (int i = start ; i < end; i++){
                copySegments(a,b,index,i);
       }// end i , alle medianer

      // ---------- sync 3 ------------------------------
       try { barrier.await();} catch(Exception e) {return;}

       // steg d)
        copyBack(a,b,start,end-1);
```

```
// (run forts.)
// ---------- sync 4 ------------------------------
  try { barrier.await();} catch(Exception e) {return;}

     // steg e)
      if (ind ==0)   insertSort(a);
} // end run
```

Oppgaver:

7.1 Programmer denne algoritmen sekvensielt.

7.2 Lag en parallell versjon av dette sekvensielle programmet med Modell2 koden, bare parallelliser stegene a) , c) og d) - ikke stegene b) eller e) som begge er kall på innstikksortering. Skriv bare de tilleggene du trenger og forklar hvor du plasserer dem i Modell2-koden,