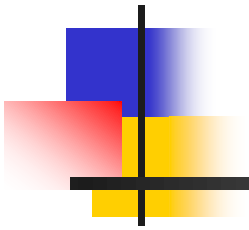


INF2440 Uke 10, v2014 :

Arne Maus
OMS,
Inst. for informatikk

Hva så vi på i uke 9

- Et sitat om tidsforbruk ved faktorisering
- En presisering av Oblig2.
- Om en feil i Java ved tidtaking (tid == 0 ??)
- Hvor lang tid de ulike mekanismene vi har i Java tar.
- Hvordan parallellisere rekursive algoritmer
- Gå IKKE i 'direkte oversettelses-fella'
  - eksemplifisert ved Kvikk-sort, 3 ulike løsninger

Hva skal vi se på i Uke10

- En kommentar om Oblig2
- Automatisk parallellisering av rekursjon
- PRP- Parallel Recursive Procedures
  - Nåværende løsning (Java, multicore CPU, felles hukommelse) – implementasjon: Peter L. Eidsvik
  - Mange tidligere implementasjoner fra 1994
    - (C over nettet, C# på .Net, Java over nettet,..)
- Demo av to kjøringer
- Hvordan ikke gå i fella: Et Rekursivt kall = En Tråd
  - Halvautomatisk
- Hvordan kan vi bygge en kompilator (preprosessor) for automatisk parallellisering
  - Prinsipper (bredde-først og dybde-først traversering av r-treet)
  - Datastruktur
  - Eksekvering
- Krav til et program som skal bruke PRP

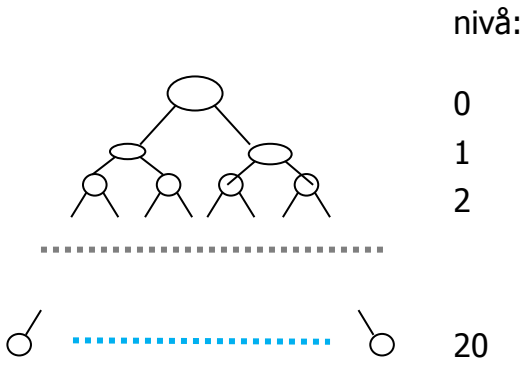
Generelt om rekursiv oppdeling av a[] i to deler

```
void Rek (int [] a, int left, int right) {
        <del opp omradet a[left..right] >
        int deling = partition (a, left,right);

        if (deling - left  > LIMIT ) Rek (a,left,deling-1);
        else <enkel løsning>;
        if (right - deling > LIMIT) Rek (a,deling, right);
        else  <enkel løsning>
}
```

```
void Rek(int [] a, int left, int right) {
        <del opp omradet a[left..right]>
        int deling = partition (a, left,right);
        Thread t1 = null, t2= null;

        if (deling - left > LIMIT )  t1 = new Thread (a,left,deling-1);
        else <enkel løsning>;
        if (right - deling > LIMIT) t2 = new Thread (a,deling, right);
        else  <enkel løsning>
        try{ if (t1!=null)t1.join();
              if (t2!=null)t2.join();} catch(Exception e){};
}
```

Hvor mange kall gjør vi i en rekursiv løsning?

- Anta Quicksort av n $=2^k$ tall
  (k= 10 $\Rightarrow$ n = 1000, k= 20 $\Rightarrow$ n= 1 mill)
- Kalltreet vil på første nivå ha 2 lengder av $2^{19,}$ på neste: 4 = 2^2 hver med 2^{18} og helt ned til nivå 20, hvor vi vil ha 2^{20} kall hver med 1 = 2^0 element.
- I hele kalltreet gjør vi altså 2 millioner -1 kall for å sortere 1 mill tall !
- Bruker vi innstikksortering for n < 32 = 2^5 så får vi 'bare' $2^{20-5} = 2^{15}$ = 32 768 kall.
- Metodekall tar : **2-5** μs og kan også optimaliseres bort (og gis speedup >1)
- **Å lage en tråd og starte den opp tar: ca.1500 μs, men ca. 180 μs for de neste trådene (med start())**



Vi kan IKKE bare erstatte rekursive kall med nye tråder i en rekursiv løsning !

Konklusjon om å parallellisere rekursjon

- Antall tråder må begrenses !
- I toppen av treet brukes tråder (til vi ikke har flere og kanskje litt mer)
- I resten av treet bruker vi sekvensiell løsning i hver tråd!
- Viktig også å kutte av nedre del av treet (her med insertSort) som å redusere treets størrelse drastisk (i antall noder)
- Vi har for n = 100 000 gått fra:

n	sekv.tid(ms)	para.tid(ms)	Speedup	
100000	34.813	41310.276	0.0008	Ren trådbasert
100000	8.118	823.432	0.0099	Med insertSort
100000	7.682	5.198	1.4777	+ Avkutting i toppen

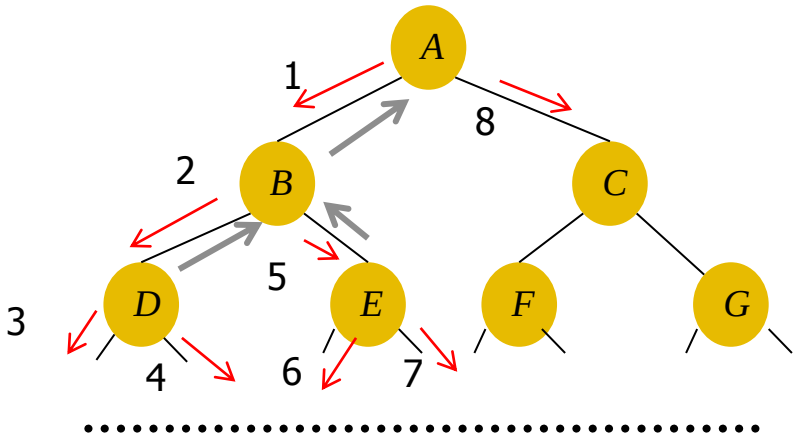
Speedup > 1 og **ca. 10 000x fortere enn ren oversettelse**.

Drømmen om lage automatisk parallelliseing

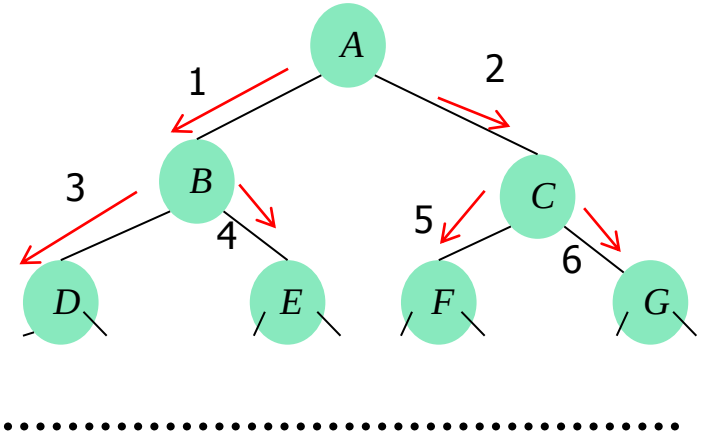
- Parallelliseing gir lang og vanskelig å lage kode
- Det finnes særlig to typer av sekvensielle programmer som tar 'lang' tid:
  - A) Med løkker (enkle, doble,..)
  - B) Rekursive
- Drømmen er man bare helt automatisk, eller bare med noen få kommandoer kan oversette et sekvensielt program til et parallelt.
  - Med løkker hadde vi bl.a HPFortran (Fortran90) som paralleliserte løkker (slo ikke helt an)
  - Reursive metoder – vi skal se på PRP (et system jeg har fått lager som hovedfagsoppgaver flere ganger siden ca. 1995)

PRP: Den grunnleggende ideen:
– Rekursjonstreet og treet av tråder er 'like', men

Rekursjon



Tråder



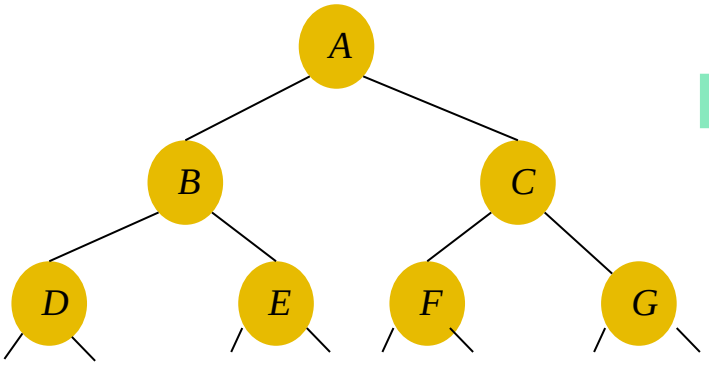
Dybde først

Bredde først

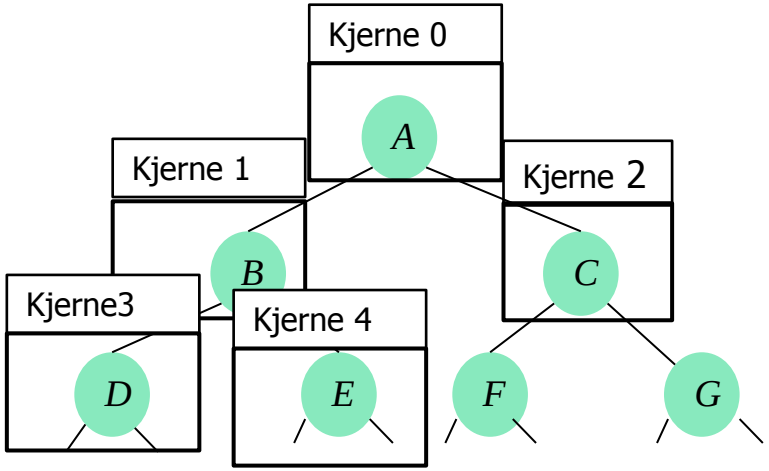
Tråder:
- Kan ikke bruke 'svaret' fra venstre-tråd til å lage parametere for høyre-tråd
- Venter på trådene (og 'svaret') først når begge trådene er sendt ut..

Fra rekursjon til parallelle prosedyrer (metoder)

Rekursjon

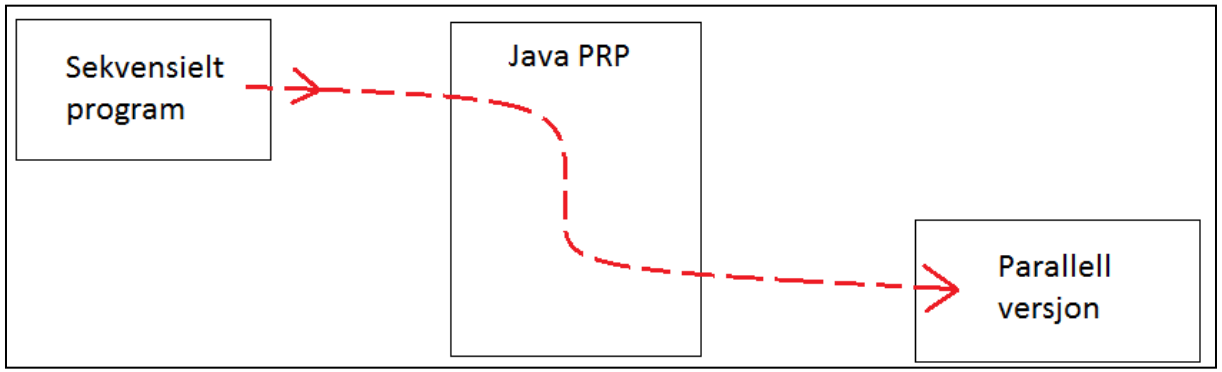


Multikjerne CPU



I Uke 9 så vi på å overføre Rekursjon til tråder - fra Peter Eidsviks (kommende) masteroppgave

- Vi skal nå automatisere det
- Vi lager en preprosessor: javaPRP
  - dvs. et Java-program som leser et annet Java-program og omformer det til et annet, gyldig Java-program (som er det parallelliserte programmet med tråder)



- For at JavaPRP skal kunne gjøre dette, må vi legge inn visse kommentarer i koden:
  - Hvor er den rekursive metoden
  - Hvor er de rekursive kallene
- Bare rekursive metoder med to eller flere kall, kan paralllelliseres .

Et eksempel før mer 'teori' med en kjørbar sekvensiell Quicksort

```
import java.util.Random;

class QuicksortProg{
    public static void main(String[] args){
        int len = Integer.parseInt(args[0]);
        int [] tid = new int[11];
        for(int i = 0; i < 11; i++){
            int[] arr = new int[len];
            Random r = new Random();
            for(int j = 0; j < arr.length; j++){
                arr[j] = r.nextInt(len-1);
            }
            long start = System.nanoTime();
            int[] k =
         new QuicksortCalc().quicksort (arr,0,arr.length-1);
            long timeTakenNS = System.nanoTime() - start;
            tid[i] = (int) timeTakenNS/1000000;
            System.out.println(timeTakenNS/100000.0);
        }
        tid = QuicksortCalc.insertSort(tid,0,10);
        System.out.println("Median sorteringstid for 11
                    gjennomlop:"+tid[5]+"ms. for n="+len);
    }
}
```

```
class QuicksortCalc{
    int INSERT_LIM = 48;
    int[] quicksort (int[] a, int left, int right){
        if (right-left < INSERT_LIM){
            return insertSort(a,left,right);
        }else{
            int pivotValue = a[(left + right) / 2];
            swap(a, (left + right) / 2, right);
            int index = left;

            for (int i = left; i < right; i++) {
                if (a[i] <= pivotValue) {
                    swap(a, i, index);
                    index++;
                }
            }
            swap(a, index, right);
            int index2 = index;
            while(index2 > left && a[index2] == pivotValue){
                index2--;
            }
           a = quicksort (a, left,  index2);
           a = quicksort (a, index + 1, right);
            return a;
    }   }
```

ok pakket den inn i en klasse

ganger)

QuickSort av 10 mill tall (ca. 0.95 sek) sekvensielt

```
M:\INF2440Para\PRP>java QuicksortProg 10000000
919.320237
948.897802
950.035171
946.001883
937.006513
940.43017
1027.33572
995.356381
1011.87974
934.688378
957.091764
Median sorteringstid for 11 gjennomlop:948ms. for n=10000000
```

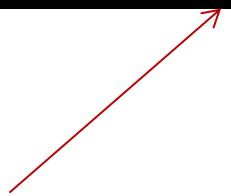
Nå legger vi til tre kommentarer så den kan preprosessereres over i en parallell versjon (/*REC*/ og /*FUNC*/):

```
import java.util.Random;

class QuicksortProg{
    public static void main(String[] args){
        int len = Integer.parseInt(args[0]);
        int [] tid = new int[11];
        for(int i = 0; i < 11; i++){
        for(int i = 0; i < 11; i++){
            int[] arr = new int[len];
            Random r = new Random();
            for(int j = 0; j < arr.length; j++){
                arr[j] = r.nextInt(len-1);
            }
            long start = System.nanoTime();
            int[] k =
             new QuicksortCalc().quicksort(arr,0,arr.length-1);
            long timeTakenNS = System.nanoTime() - start;
             tid[i] = (int) timeTakenNS/1000000;
            System.out.println(timeTakenNS/100000.0);
        }
        tid = QuicksortCalc.insertSort(tid,0,10);
        System.out.println("Median sorteringstid for 11
                    gjennomlop:"+tid[5]+"ms. for n="+len);
    }
}
```

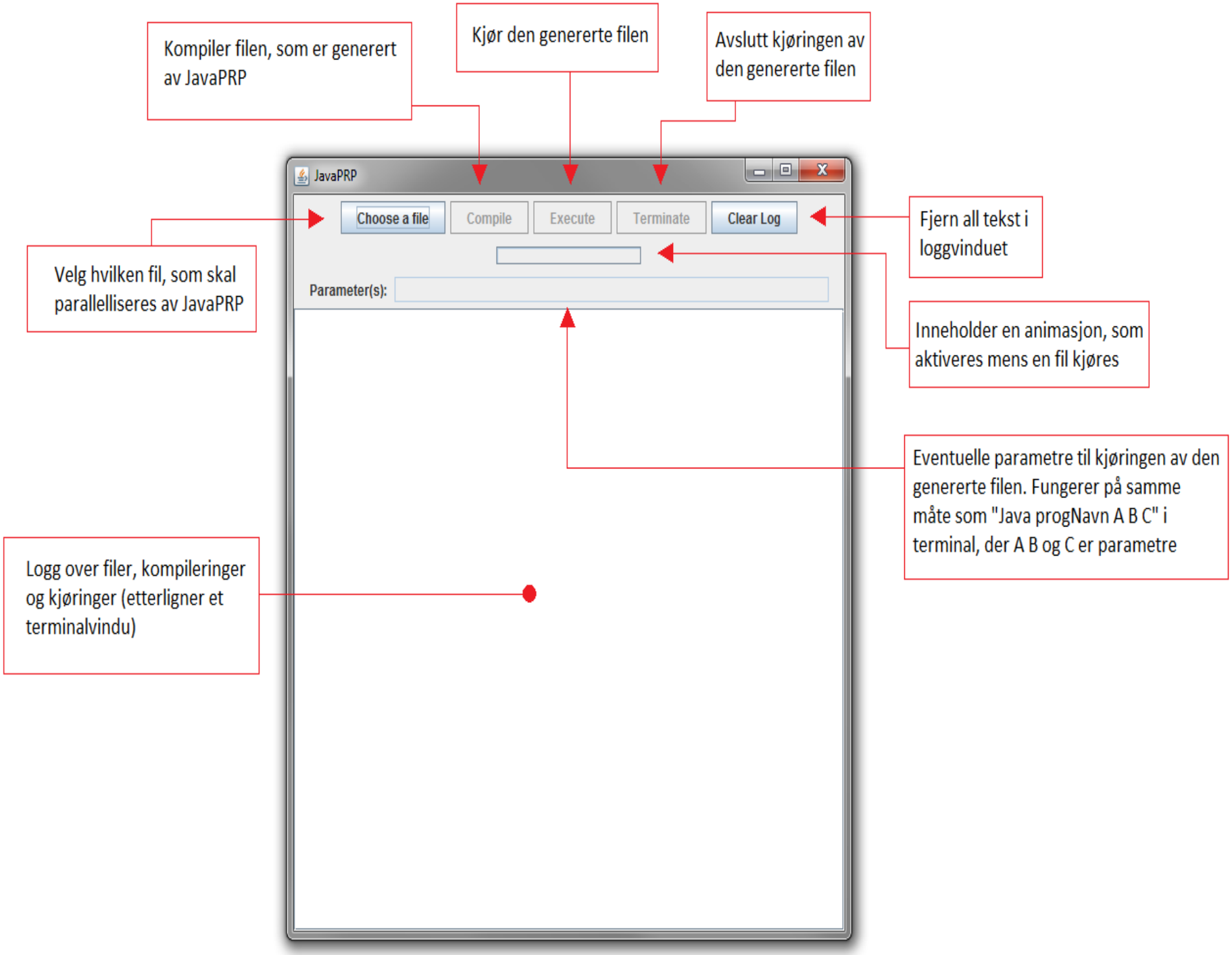
```
class QuicksortCalc{
    int INSERT_LIM = 48;
    /*FUNC*/
    int[] quicksort(int[] a, int left, int right){
        if (right-left < INSERT_LIM){
            return insertSort(a,left,right);
        }else{
            int pivotValue = a[(left + right) / 2];
            swap(a, (left + right) / 2, right);
            int index = left;

            for (int i = left; i < right; i++) {
                if (a[i] <= pivotValue) {
                    swap(a, i, index);
                    index++;
                }
            }
            swap(a, index, right);
            int index2 = index;
            while(index2 > left && a[index2] == pivotValue){
                index2--;
            }

            /*REC*/
            a = quicksort(a, left,  index2);
            /*REC*/
            a = quicksort(a, index + 1, right);
            return a;
        }
    }
```

Kompilér JavaPRP -systemet, så start det

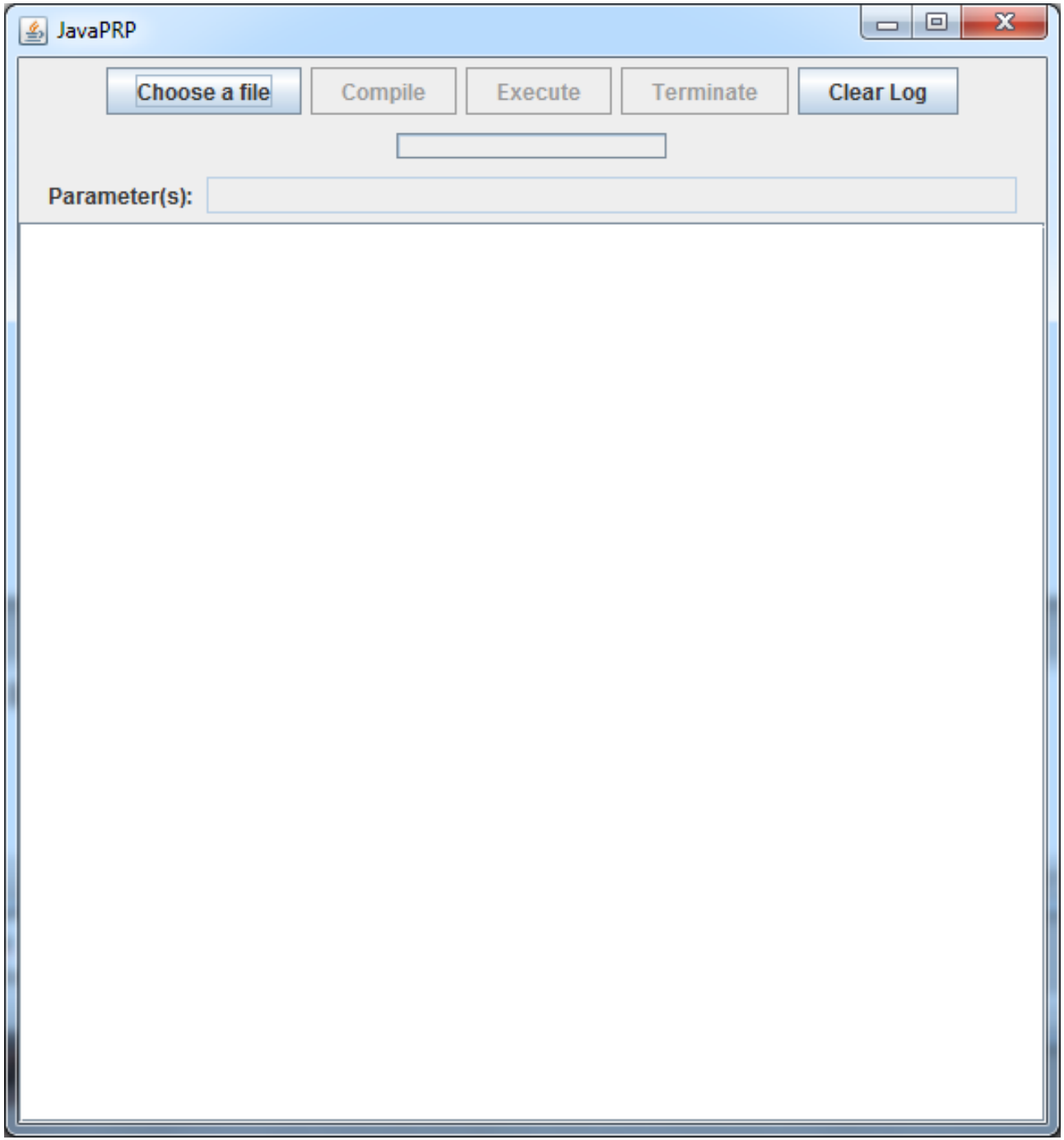
```
M:\INF2440Para\PRP>javac JavaPRP.java

M:\INF2440Para\PRP>java JavaPRP
```

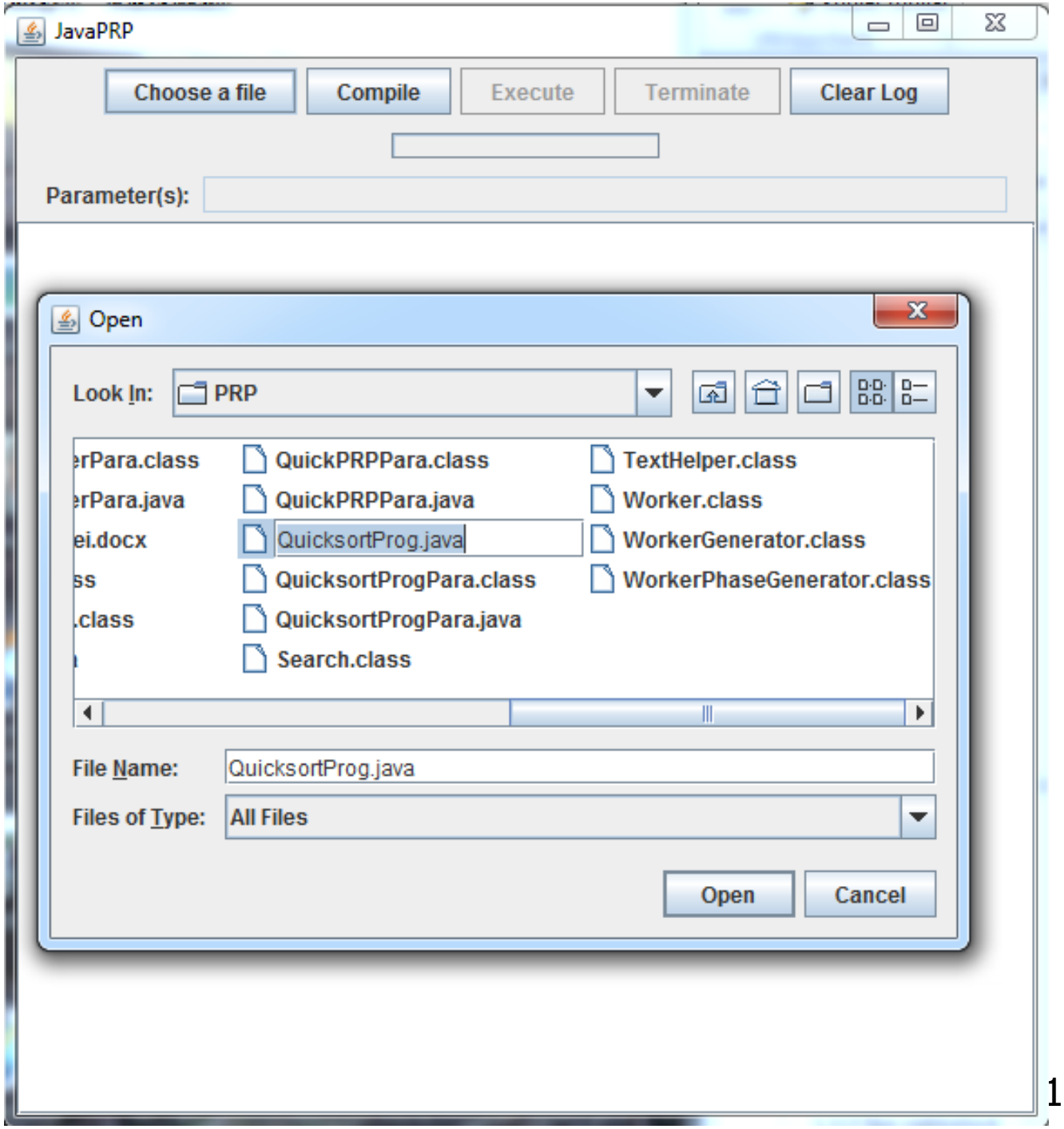


Det starter et GUI-interface


Kompiler filen, som er generert av JavaPRP
Kjør den genererte filen
Avslutt kjøringen av den genererte filen
JavaPRP
Choose a file
Compile
Execute
Terminate
Clear Log
Parameter(s):
Velg hvilken fil, som skal parallelliseres av JavaPRP
Fjern all tekst i loggvinduet
Inneholder en animasjon, som aktiveres mens en fil kjøres
Eventuelle parametre til kjøringen av den genererte filen. Fungerer på samme måte som "Java progNavn A B C" i terminal, der A B og C er parametre
Logg over filer, kompileringer og kjøringer (etterligner et terminalvindu)

Trykker: Choose a file

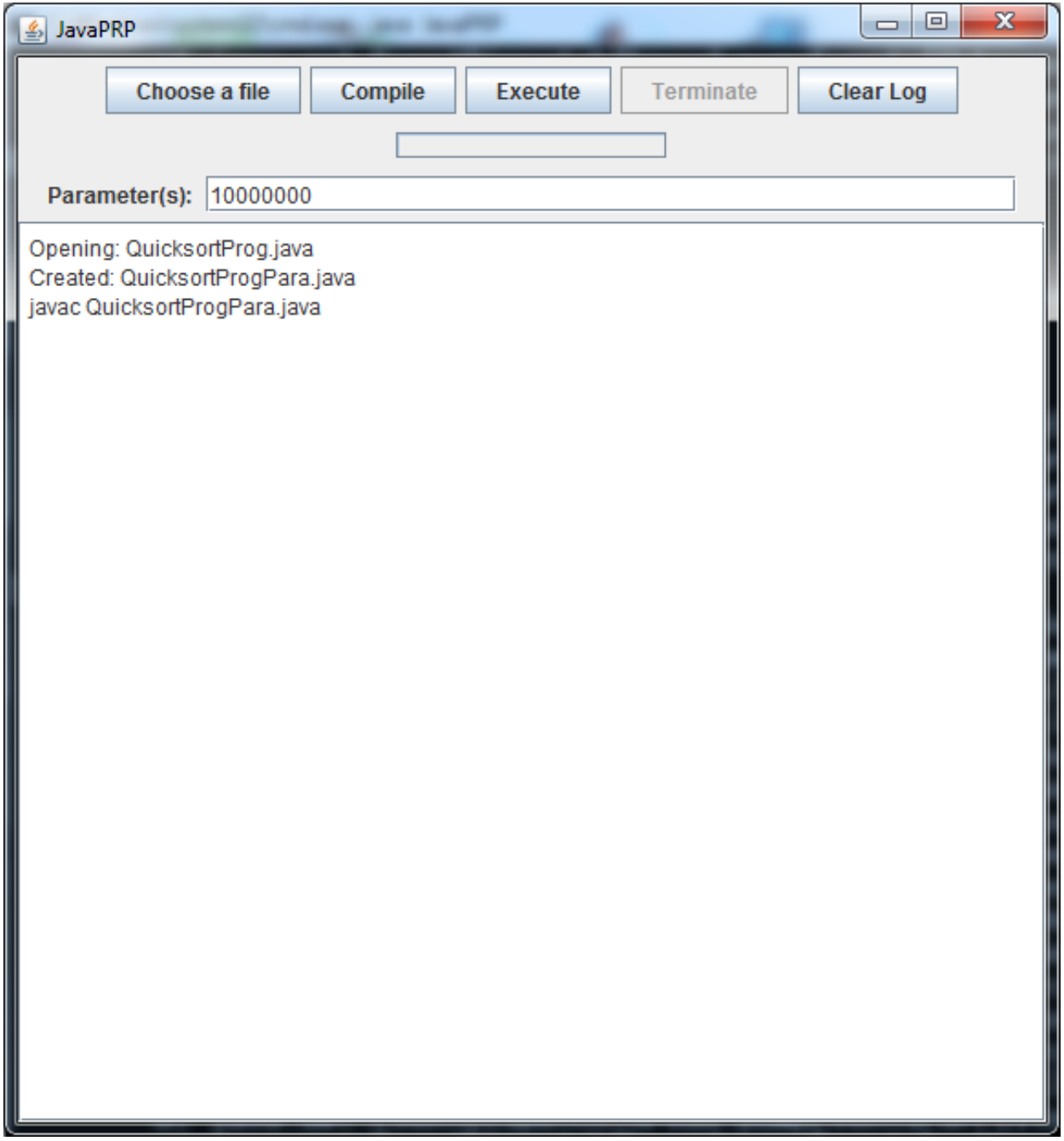



JavaPRP
Choose a file
Compile
Execute
Terminate
Clear Log
Parameter(s):
Open
Look In:
PRP
erPara.class
erPara.java
ei.docx
ss
.class
QuickPRPPara.class
QuickPRPPara.java
QuicksortProg.java
QuicksortProgPara.class
QuicksortProgPara.java
Search.class
TextHelper.class
Worker.class
WorkerGenerator.class
WorkerPhaseGenerator.class
File Name:
QuicksortProg.java
Files of Type:
All Files
Open
Cancel

Trykket så Compile:

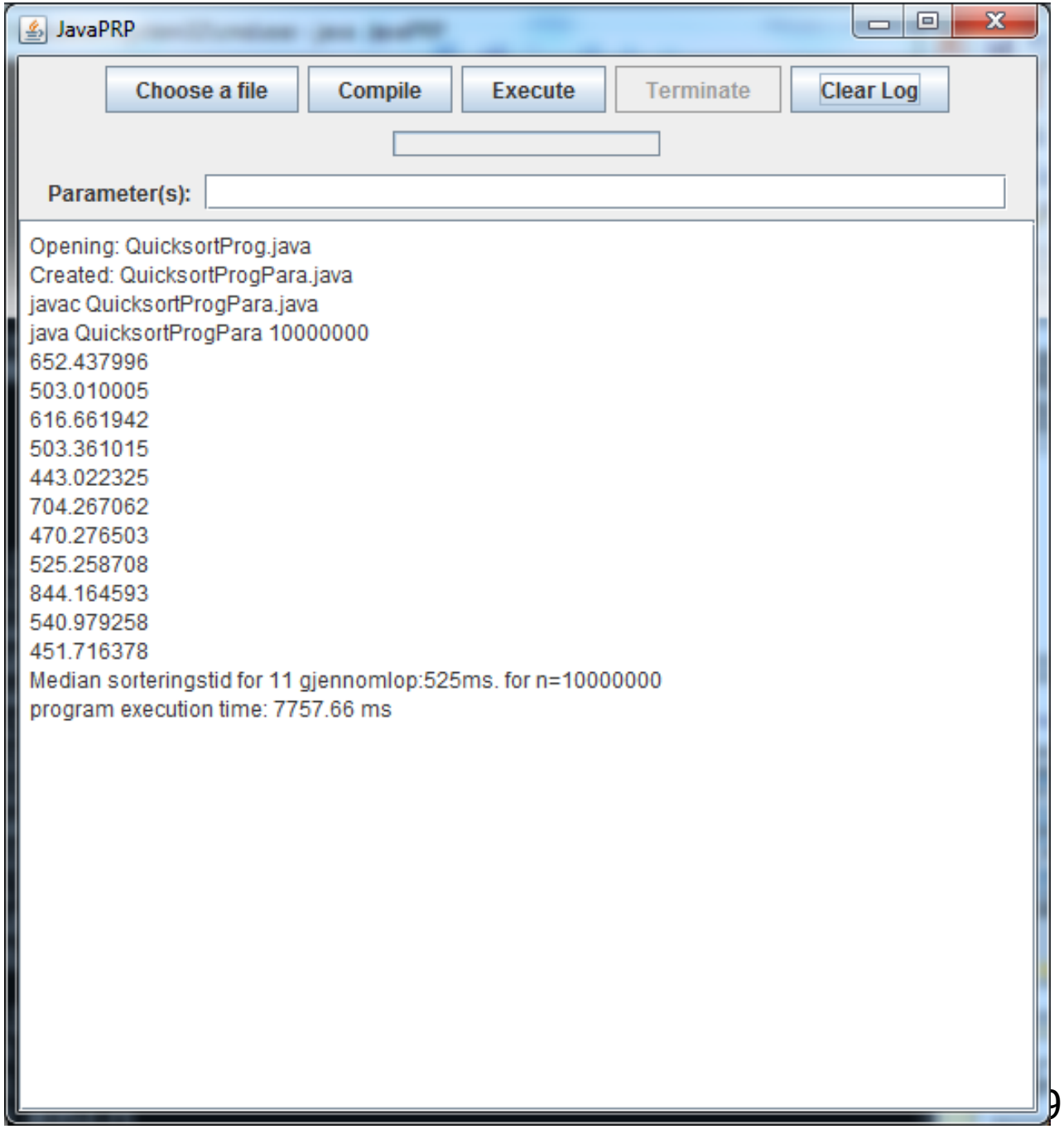
- Kompilerte da den parallelliserte filen: QuicksortProg**Para**.java

- Legger så inn parameter (10 mill) på kommandolinja og velger Execute.



Resultatet kommer i log-vinduet.
-en OK speedup for den parallelle utførelsen:

S=948/525= 1,80



Oversettelsen : Quicksort

```
import java.util.Random;

class QuicksortProg{
    public static void main(String[] args){
        int len = Integer.parseInt(args[0]);

        for(int i = 0; i < 11; i++){
            int[] arr = new int[len];
            Random r = new Random();
            for(int j = 0; j < arr.length; j++){
                arr[j] = r.nextInt(len-1);
            }
            long start = System.nanoTime();
            int[] k =
           new QuicksortCalc().quicksort (arr,0,arr.length-
            long timeTakenNS = System.nanoTime() - start
            System.out.println(timeTakenNS/100000.0);

        }
    }
}
```

Starten på brukerens kode (77 linjer)

```
class QuicksortProgPara{
        public static void main(String[] args){
                new Admin(args);
        }
}

class Admin{
    public Admin(String[] args){
        initiateParallel(args);
    }

    void initiateParallel(String[] args){
        int len = Integer.parseInt(args[0]);

        for(int i = 0; i < 11; i++){
        int[] arr = new int[len];
        Random r = new Random();
        for(int j = 0; j < arr.length; j++){
            arr[j] = r.nextInt(len-1);
        }
        long start = System.nanoTime();
        int[] k  = startThreads (arr,0,arr.length-1);
        long timeTakenNS = System.nanoTime() - start
        System.out.println(timeTakenNS/100000.0);
    }
}
```

Starten på oversatt kode (245 linjer)

Dette kan også kjøres delvis linjeorientert

Din sekvensielle rekursive løsning (som er annotert for PRP) heter MittProg.java

1) Hvis du ønsker det, kjør ditt egent program og notere eksekveringstiden.

    >javac MittProg.java

    >java MittProg 1000000

2) Kompilér PRP-systemet (hvis du ikke har gjort det tidligere)

    >java javaPRP.java

3) Oversett ditt program (MittProg.java) til et parallelt program (MittProgPara.java) – dette må gjøres via GUI

    >java javaPRP – velg da MittProg.java og trykk Compile

4) Kjør det genererte parallelle programmet (MittProgPara.java)

    > java MittProgPara 1000000

PRP kan også kan parallelliseres et fasedelt program

- Et fasedelt PRP-program har flere faser som hver består av først en parallell rekursiv del og så en sekvensiell del (kan sløyfes).
- Da må brukeren beskrive det i en egen ADMIN –metode:

```
/*ADMIN*/
public int minAdminMetode(...){
    int svar = rekursivMetode1(...);
    sekvensiellKode1();
    svar = rekursivMetode2(...);
    sekvensiellKode2(...);
    svar = rekursivMetode3(...);
    sekvensiellKode3(...);
    svar = rekursivMetode4(...);
    sekvensiellKode4(...);
    return svar;
}
```

- Innfører da to nye Kommentarkoder: /*ADMIN*/ og **/*FUNC 1*/ , /*FUNC 2*/,… osv**

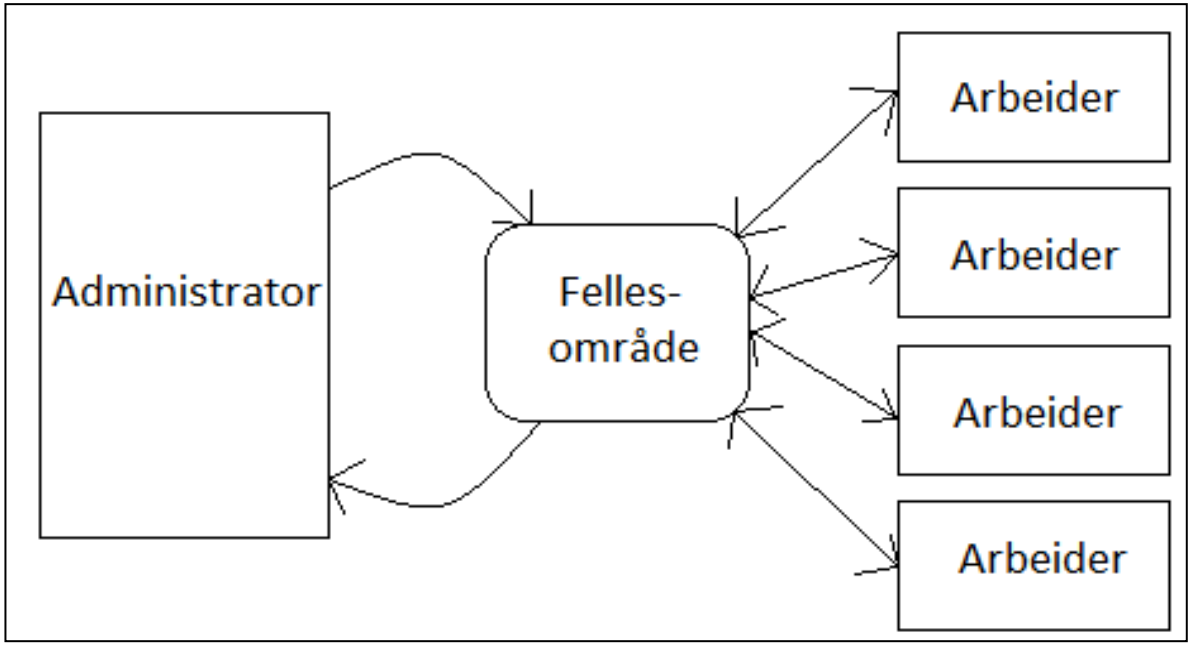
Og hver av disse rekursive metodene er i hver sin klasse.

```
class MittFaseProgram{
   public static void main(String[] args){
    <returverdi> svar =
            new MittFaseProgram().minAdminMetode(...);
   }

   /*ADMIN*/
   <returverdi> minAdminMetode(...){
      <returverdi> svar1 = new Fase1().rekursivMetode(...);
      sekvensiellKode1(...);
      <returverdi> svar2 = new Fase2().rekursivMetode(...);
      sekvensiellKode2(...);
      return ...;
   }

   void sekvensiellKode1(...){
    }

   void sekvensiellKode2(...){
   }
} // end MittFaseProgram
```

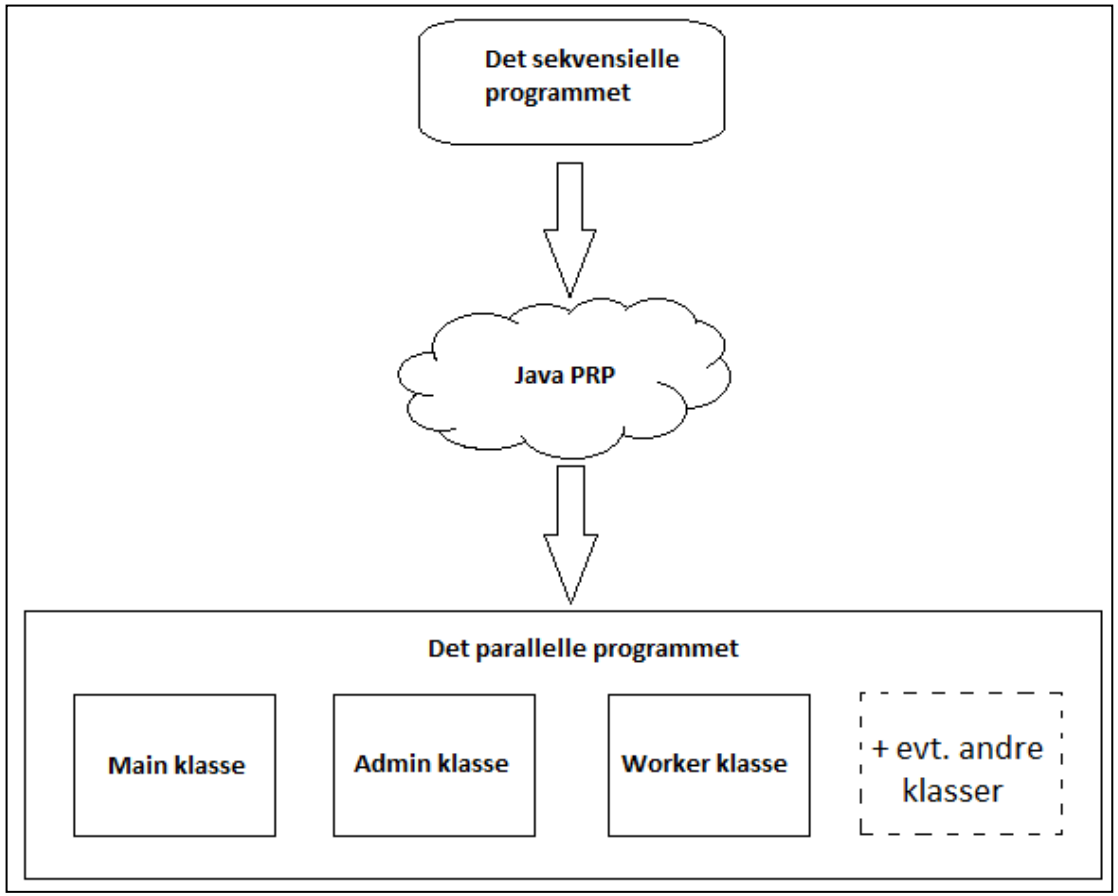
```
class Fase1{
   /*FUNC 1*/
    <returverdi> rekursivMetode(...){
      /*REC*/
      <returverdi> svar1 = rekursivMetode(...);
      /*REC*/
      <returverdi> svar2 = rekursivMetode(...);
      return ...;
   }

}

class Fase2{
   /*FUNC 2*/
 <returverdi> rekursivMetode(...){
      /*REC*/
      <returverdi> svar1 = rekursivMetode(...);
      /*REC*/
      <returverdi> svar2 = rekursivMetode(...);
      /*REC*/
      <returverdi> svar3 = rekursivMetode(...);
      return ...;
   }
}
```

Hvordan gjøres dette? Administrator – arbeider modell

- Oppgaver legges ut i et fellesområde
- Arbeiderne tar oppgaver og legger svar tilbake i fellesområdet



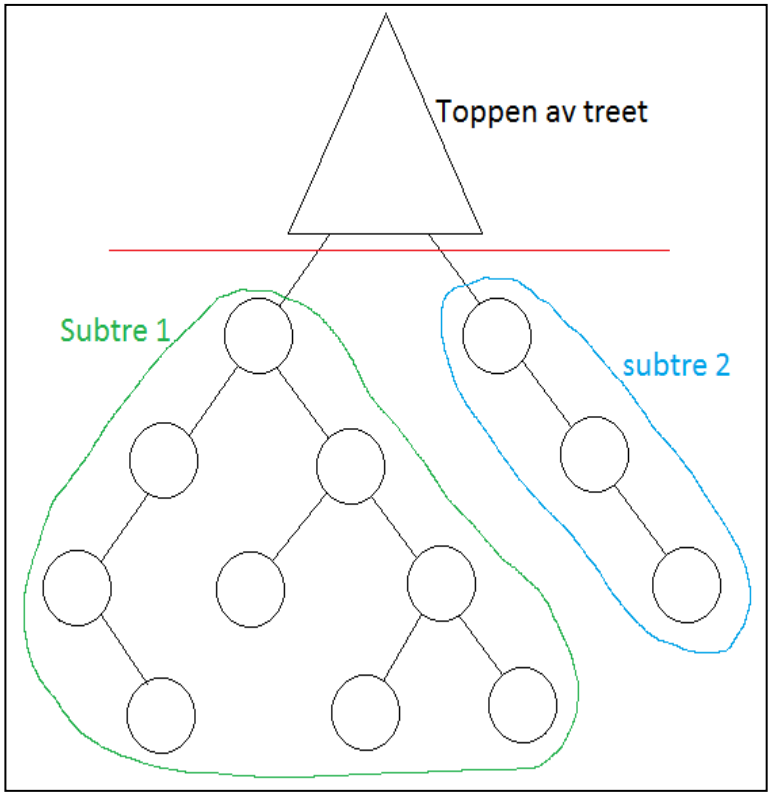
Fra sekvensielt program til parallelt:



Figur 5: *Fra det sekvensielle programmet, gjennom Java PRP og til det parallelle resultatet. Det nye, parallelle programmet vil inneholde en klasse for main, Admin, Worker pluss eventuelt andre klasser fra det sekvensielle programmet, som ikke er en del av parallelliseringen.*

Eksempel: Parallellisering med to tråder

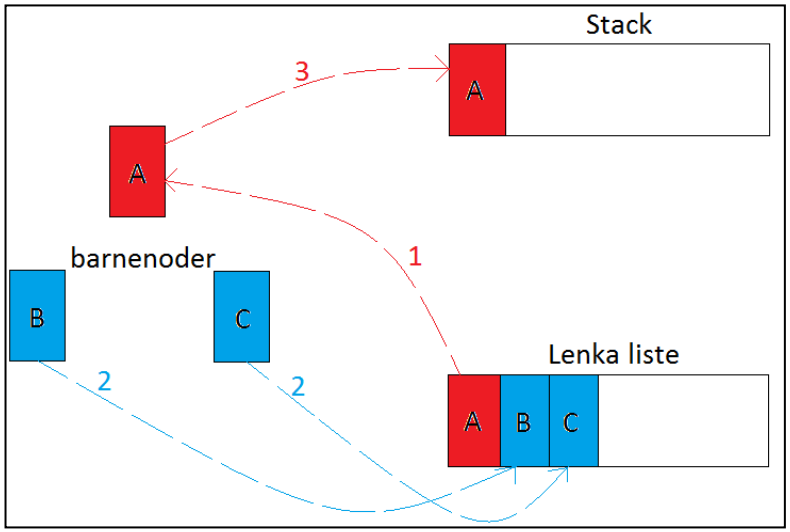
PRP: Toppen kjøres med tråder bredde-først, så går hver tråd over til dybde først og 'vanlige' rekursive kall



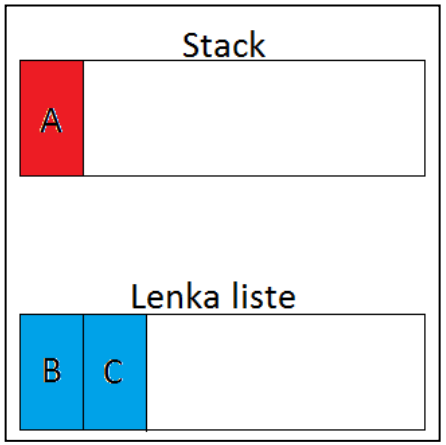
Figur 6: *Visualisering av treet der vi ønsker å parallellisere med to tråder. Toppen av treet tilsvarer bredde først traversering til vi ender på, i dette tilfelle, to subtrær. Disse to subtrærne vil traverseres dybde først*

Fra bredde først til dybde først og så vanlig rekursjon

- Trådene blir når de lages lagt i en LenkaListe (FIFO-kø)
- Så tas den første(A) i køen ut, og den lager to nye barne-tråder (B,C) som legges i lista.
- A legges i en stack (LIFO-kø)
- Neste på i Lista (B) tas ut og dens nye barne-tråder (D,E) legges inn i Lista
- B legges på stacken,… osv.
- Bunnen av bredde-først ligger da på toppen av stacken



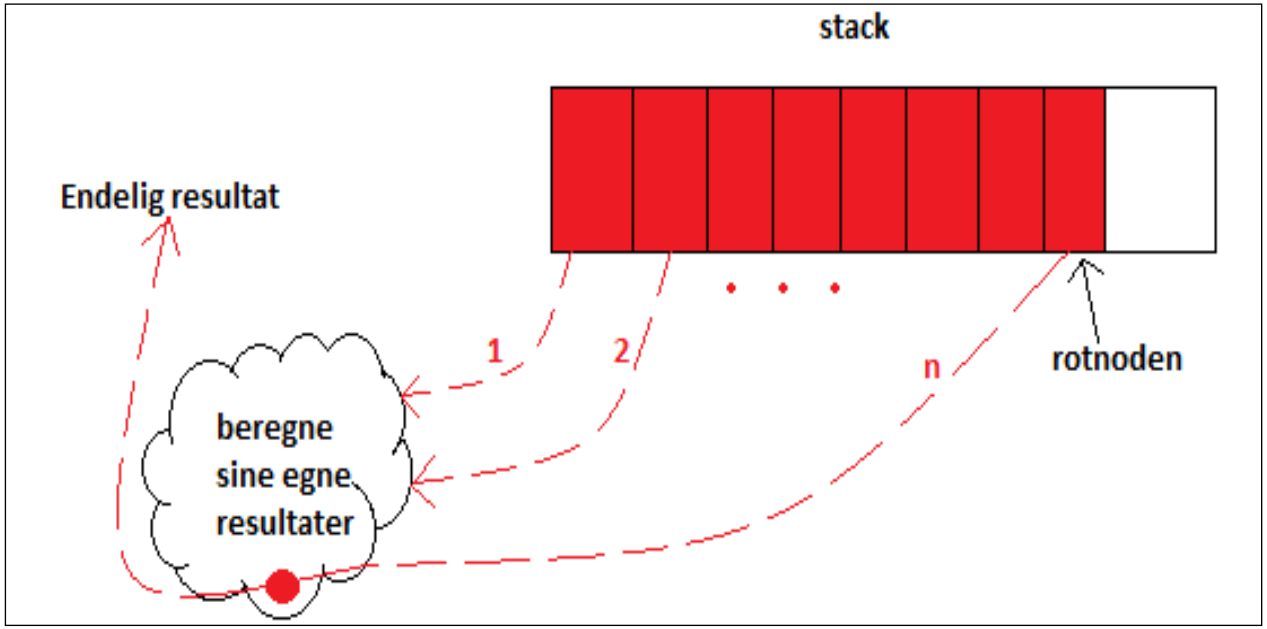
Figur 7: *Administratoren oppretter datastrukturen til arbeiderne.*



Figur 8: *Datastrukturen etter at Figur 7 er ferdig.*

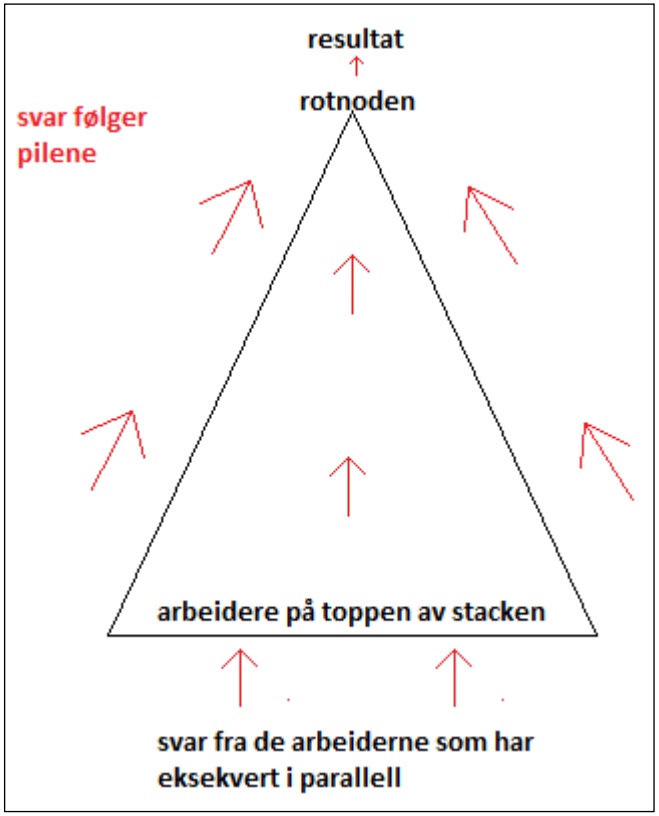
Kjøring og retur av verdier

- Når trådene er brukt opp, pop-es stacken (f.eks øverst er E) og vanlige rekursive kall gjøres fra E , neste pop-es …osv. til stacken er tom
- Svarene fra ethvert element som tas av stacken legges i en tabell og plukkes opp av den som kalte elementet.
- Den som kalte, kan så fortsette sin kode og selv returnere sitt svar,..



Figur 9: *Arbeiderstacken beregner seg innover til roten.*

Svarene genereres nederst stacken og svarene propaganderer oppover til første kall på den rekursive metoden



Hvorfor dette med bredde-og dybde-først ?

- Kunne vi ikke bare startet trådene og latt de alle gå i parallell?
- NEI – fordi:
  - Vi har lovet rekursiv semantikk (virkemåte) i den parallelle.
  - Vi skal derfor oversette det rekursive programmet slik at det gir alltid samme resultat parallelt.
- Eks Quicksort:
  - Anta at vi parallelliserer ned til nivå 3 i treet
  - Hvis nivå 2 og 3 går samtidig, vil dette gå galt fordi de prøver begge lagene og flytte på de samme elementene i a[].

Hva gjøres teknisk – vår program består av (minst) to klasser

- Klassen med 'main' og en klasse som inneholder den rekursive metoden
  - Er det flere rekursive metoder som skal parallelliseres, så skal de være inne i hver sin klasse
- Det genererte programmet består av minst følgende klasser:
  - Admin
  - Worker
- Kort og greit:
  - Det oversatte programmet 'xxxxxPara.java' skal vi egentlig ikke se på og spesielt ikke endre.
  - Det bare virker og parallelliserer etter visse prinsipper.

Eksempel 2: Største tall i en array

```
class LargestNumber{
    public static void main(String[] args){
        int len = Integer.parseInt(args[0]);
        int cores =
              Runtime.getRuntime().availableProcessors();
        int[] arr = new int[len];
        Random r = new Random();
        for(int i = 0; i < arr.length; i++){
            arr[i] = r.nextInt(len-1);
        }
         long t = System.nanoTime();
        /*CALL*/
        int k = (new
        Search()).findLargest(arr,0,arr.length);
        Double t2 =(System.nanoTime()-t)/1000000.0;
        System.out.println("Largest number is " + k+
                                  ", paa:"+t2+"ms.");    }
}
```

```
class Search{
    int k = 5;

    /*FUNC*/
    int findLargest(int[] arr, int start, int end){

        if((end-start) < k){
            return largest_baseCase(arr,start,end);
        }
        int half = (end-start) / 2;
        int mid = start + half;
        /*REC*/
        int leftVal = findLargest (arr,start,mid);
        /*REC*/
        int rightVal = findLargest (arr,mid+1,end);
        if(leftVal > rightVal) return leftVal;
        return rightVal;

    }

    int largest_baseCase(int[] arr, int start, int end){
        int largest = 0;
        for(int i = start; i < end; i++){
            if(arr[i] > largest){
                largest = arr[i];
            }
        }
        return largest;
    }
}
```

NB. for å få kjøretiden riktig for det parallelle programmet

- N.B det som oppgis som «program execution time» er med overhead fra GUI løsningen. Fra GUI-en:

```
Opening: LargestNumber.java
Created: LargestNumberPara.java
javac LargestNumberPara.java
java LargestNumberPara 100000000
Largest number is 99999994, paa:152.171677ms.
program execution time: 1511.89 ms
```

- Kjør det i linjemodus (n= 100 mill.):

```
M:\INF2440Para\PRP>java LargestNumberPara 100000000
Largest number is 99999998, paa:135.922687ms.

M:\INF2440Para\PRP>java LargestNumber 100000000
Largest number is 99999998, paa:417.98199ms.
```

Hva så vi på i Uke10

- En kommentar om Oblig2
- Automatisk parallellisering av rekursjon
- PRP- Parallel Recursive Procedures
  - Nåværende løsning (Java, multicore CPU, felles hukommelse) – implementasjon: Peter L. Eidsvik
- Demo av to eksempler kjøring
- Hvordan ikke gå i fella: Et Rekursivt kall = En Tråd
  - Halvautomatisk oversettelse (hint/kommandoer fra kommentarer)
- Hvordan virker en kompilator (preprosessor) for automatisk parallellisering
  - Prinsipper (bredde-først og dybde-først traversering av r-treet)
  - Datastruktur
  - Eksekvering
- Krav til et program som skal bruke PRP
- Slik bruker du PRP – Ukeoppgave neste uke.