Obligatorisk innlevering 23.mars

# Oppgave 1

**I)**

package no.hvl.dat102;

public class SolveTowers {

public static void main(String[] args) {

long startTid = System.nanoTime();

TowersOfHanoi towers = new TowersOfHanoi(28);

towers.solve();

System.out.println("Antall flyttinger: " + towers.flyttinger);

long slutttid = System.nanoTime();

System.out.print("kjøringen tok: " + (slutttid - startTid) + " ns");

}

}

package no.hvl.dat102;

public class TowersOfHanoi {

private int totalDisks;

int flyttinger;

public TowersOfHanoi(int disks) {

this.totalDisks = disks;

flyttinger = 0;

}

public void solve() {

moveTower(totalDisks, 1, 3, 2);

}

private void moveTower(int numDisks, int start, int end, int temp) {

if (numDisks == 1) {

moveOneDisk(start, end);

} else {

moveTower(numDisks - 1, start, temp, end);

moveOneDisk(start, end);

moveTower(numDisks - 1, temp, end, start);

}

}

private void moveOneDisk(int start, int end) {

// System.out.println("Move one disk from " + start + " to " + end);

flyttinger++;

}

}

package no.hvl.dat102;

public class SumSn {

public static int oppsum(int n) {

if (n == 1) {

return 1;

} else {

return oppsum(n - 1) + n;

}

}

// 5an-1 -6an-2 +2

public static int hvertledd(int n) {

if (n > 1) {

return 5 \* hvertledd(n - 1) - 6 \* hvertledd(n - 2) + 2;

} else if(n == 1) {

return 5;

} else {

return 2;

}

}

public static void main(String[] args) {

System.out.println(oppsum(100));

System.out.println("----------------------");

// test hvert ledd test

for (int i = 0; i < 10; i++) {

System.out.println(hvertledd(i));

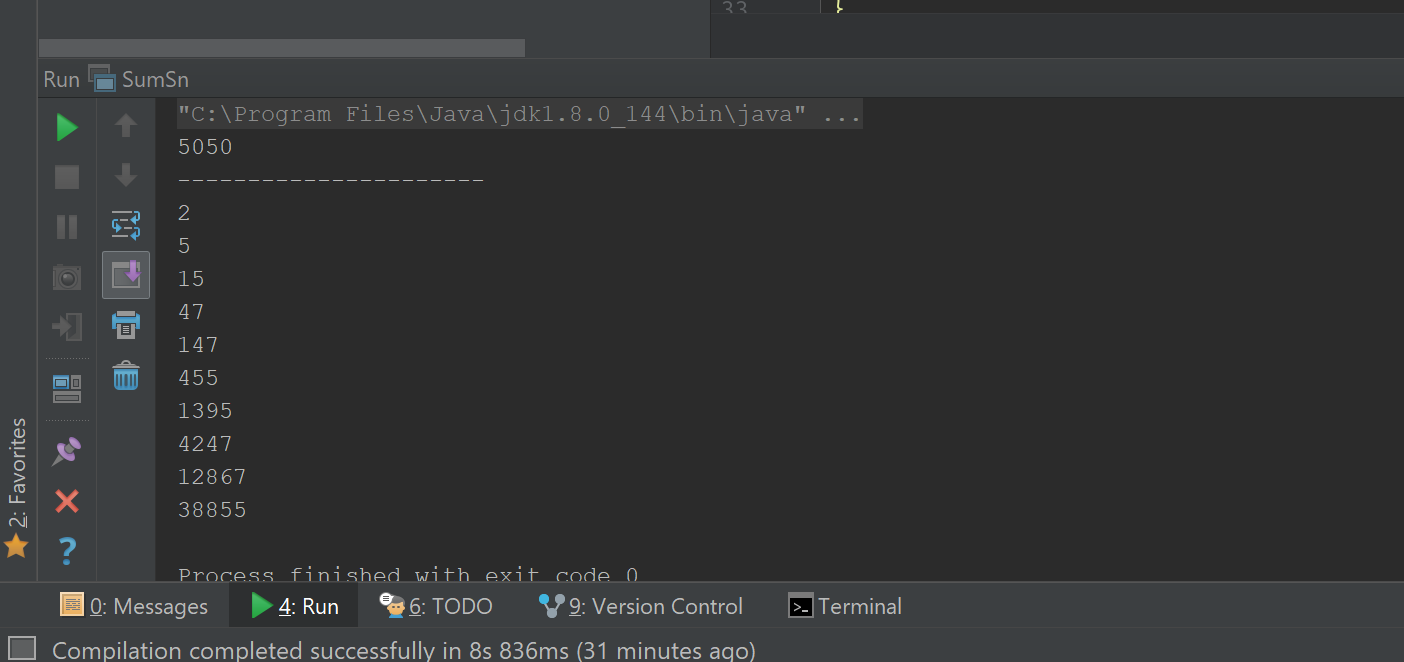
}

}

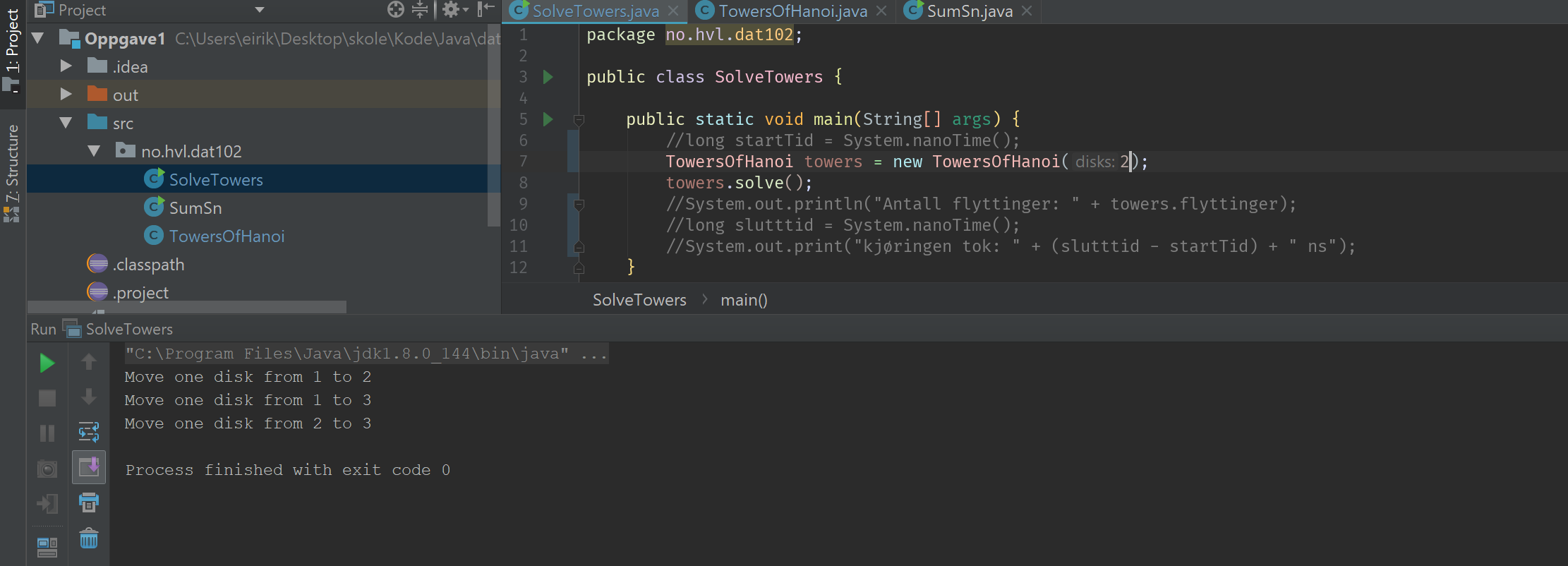
}

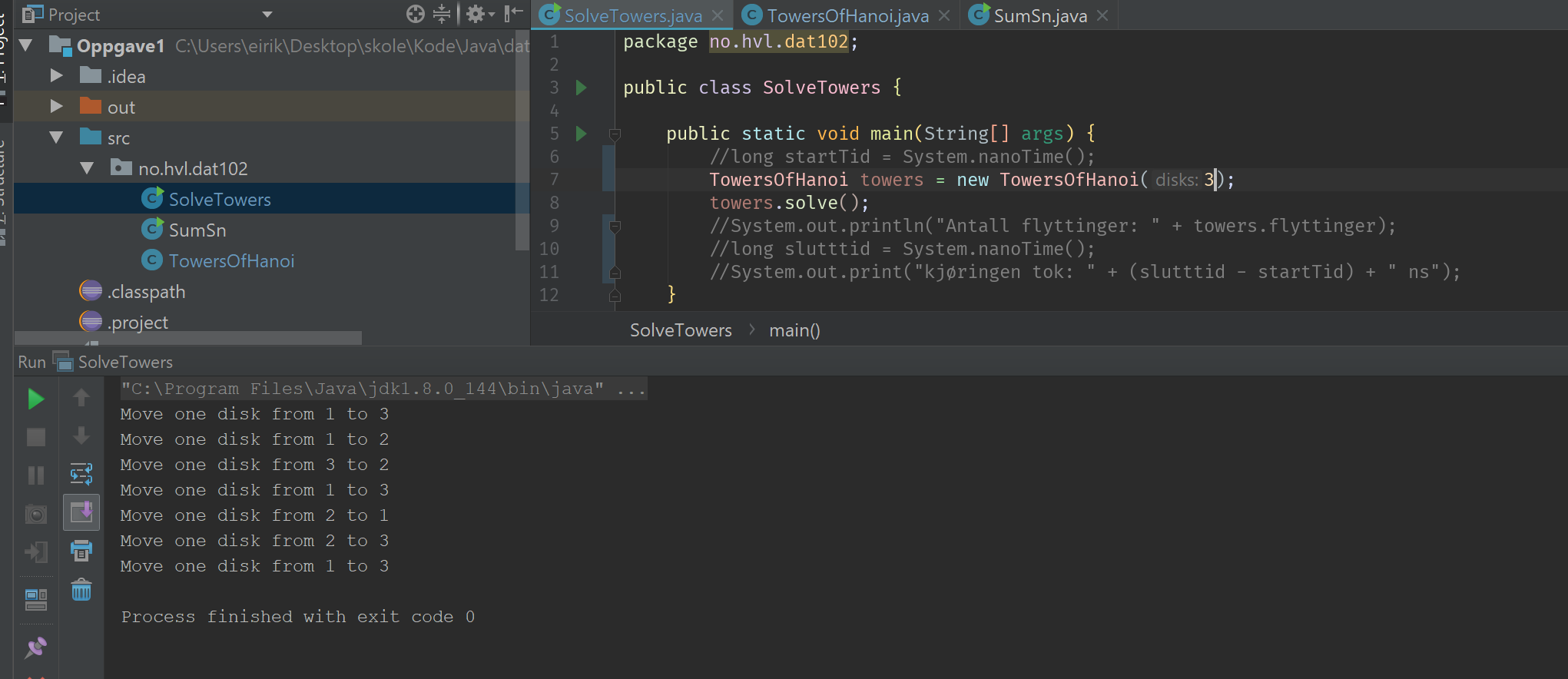
UTSKRIFT

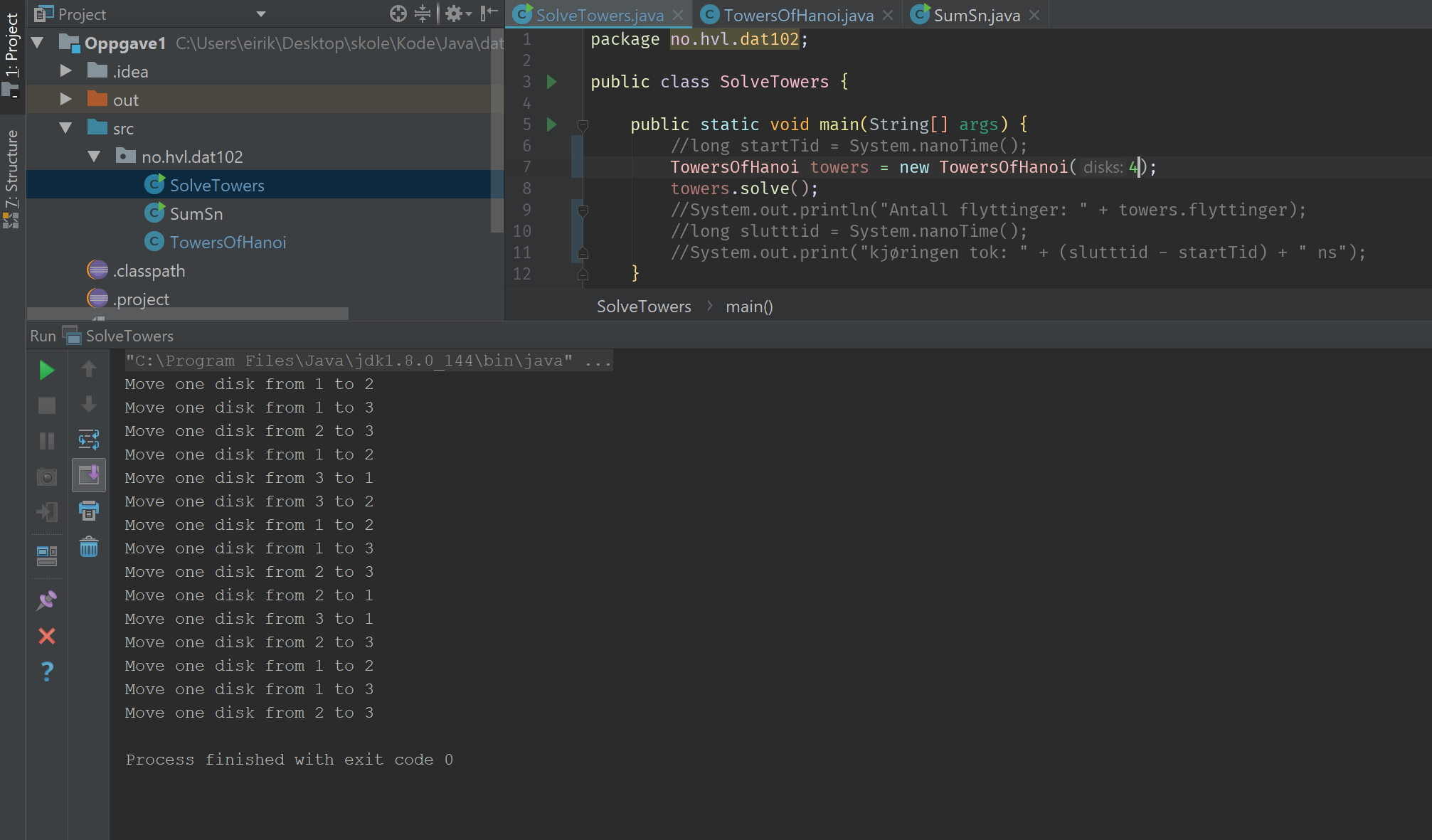
A & B:

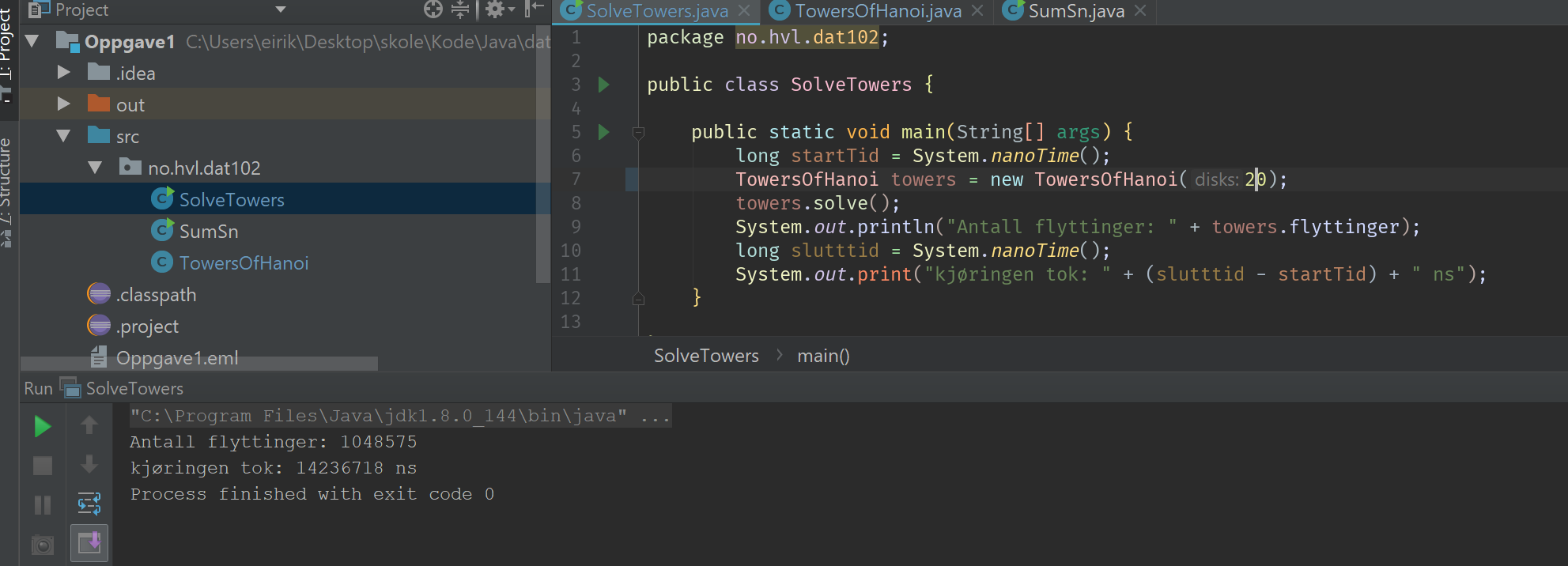


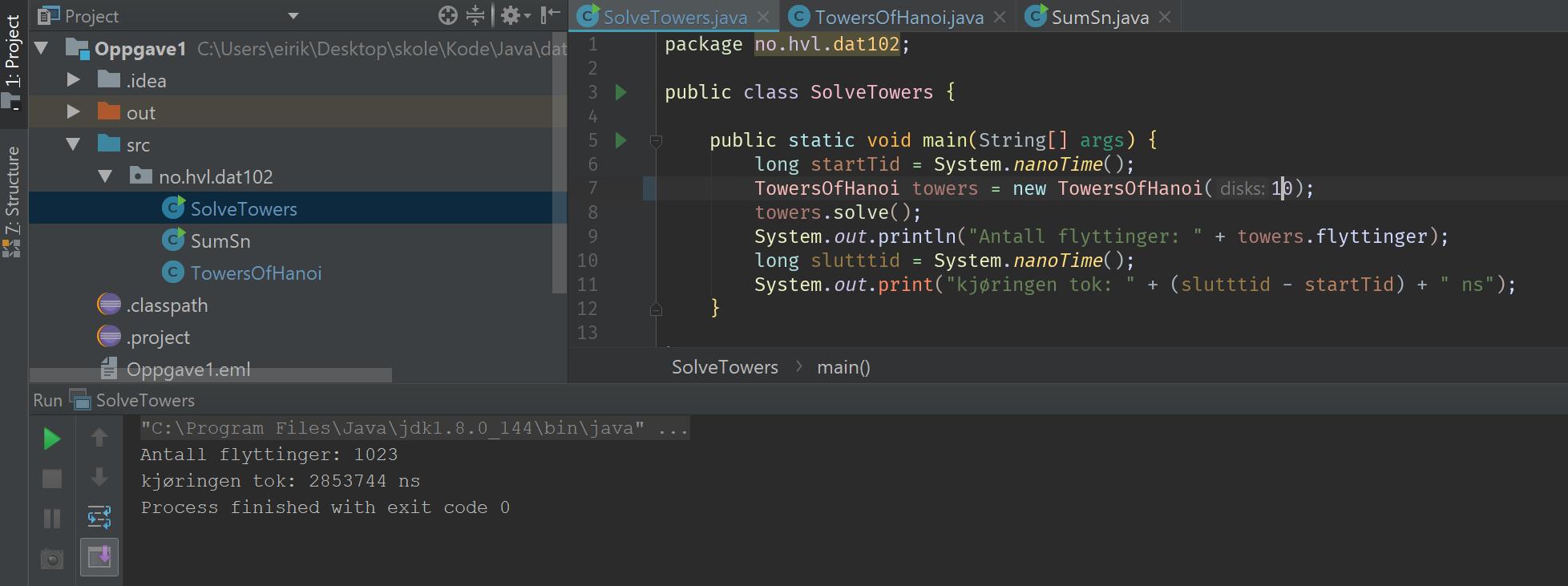
C:

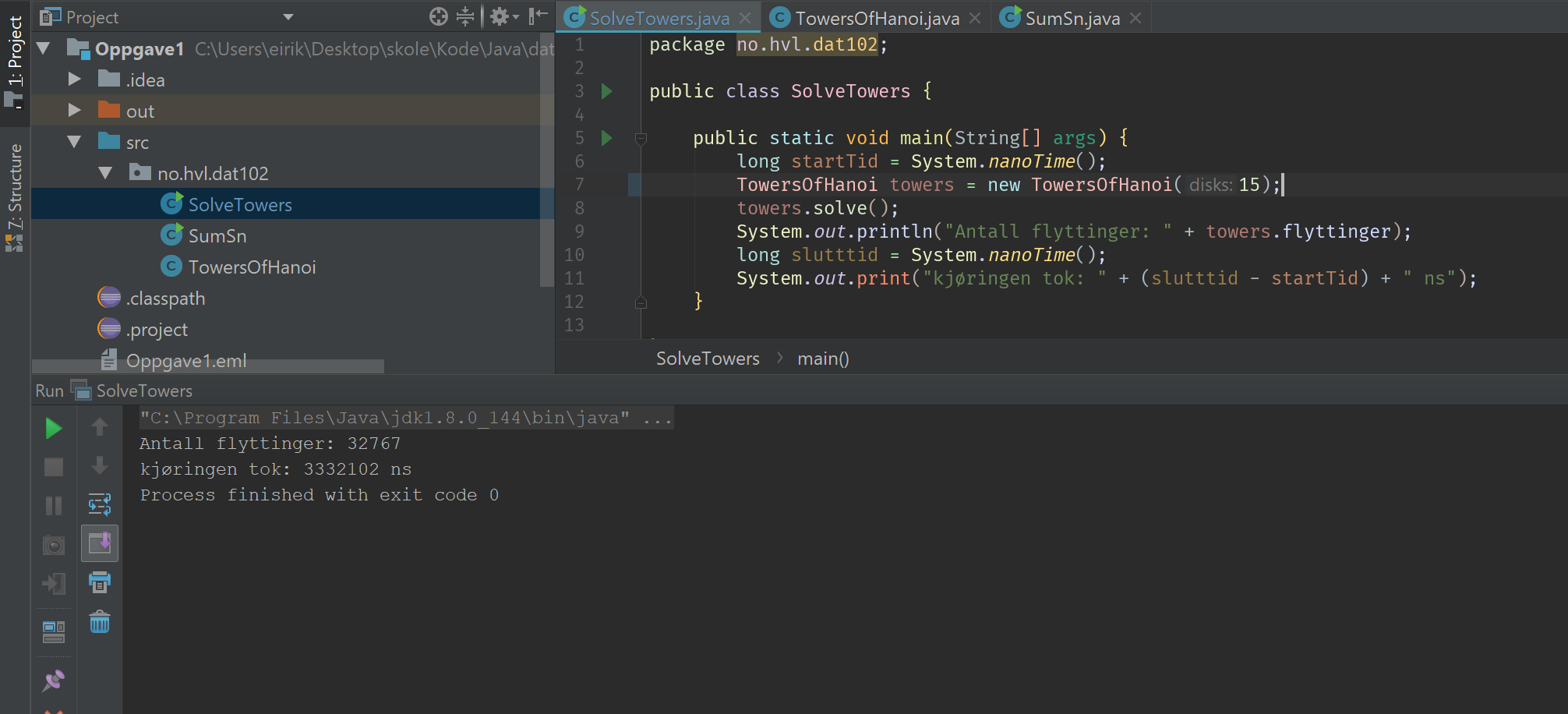


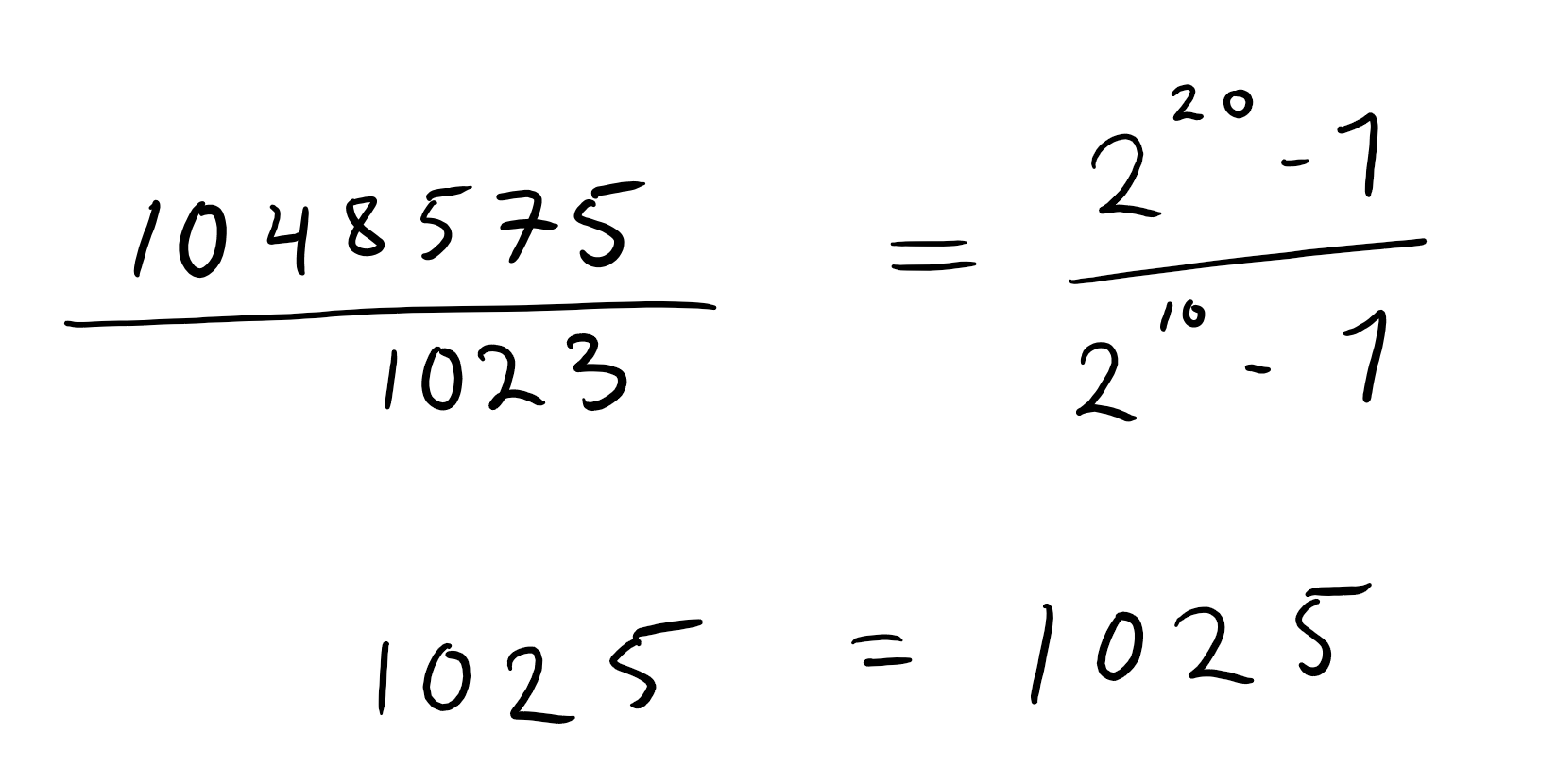












# Oppgave 2

package no.hvl.dat102.dobbelkjedetordnetlistem;

import no.hvl.dat102.adt.DobbelKjedetOrdnetListeMADT;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Representerer en dobbeltkjedet ordnet liste med midtpeker.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

public class DobbelKjedetOrdnetListeM<T extends Comparable<T>> implements DobbelKjedetOrdnetListeMADT<T> {

//M for midtpeker

private DobbelNode<T> foerste;

private DobbelNode<T> midten;

private DobbelNode<T> siste;

private int antall;

private int Ahoyre;

private int Avenstre;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Oppretter en tom liste.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Konstrukt�r

public DobbelKjedetOrdnetListeM(T minVerdi, T maksVerdi) {

// F�rste node

DobbelNode<T> nyNode1 = new DobbelNode<T>(minVerdi);

foerste = nyNode1;

midten = foerste;

// Siste node

DobbelNode<T> nyNode2 = new DobbelNode<T>();

nyNode2.setElement(maksVerdi);

nyNode1.setNeste(nyNode2);

nyNode2.setForrige(nyNode1);

siste = nyNode2;

Ahoyre = 0;

Avenstre = 0;

antall = 0;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*

// \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Override

public void leggTil(T el) {

// Setter inn ordnet f�r den noden p peker p�

DobbelNode<T> p;

if ((el.compareTo(foerste.getElement()) <= 0) || (el.compareTo(siste.getElement()) >= 0)) {

// Ugyldig. Alternativt kan vi ha det som et forkrav!

System.out.println("Ugyldig verdi. verdi > " + foerste.getElement() + "verdi < " + siste.getElement());

} else { // Kun lovlige verdier

antall++;

if (el.compareTo(midten.getElement()) >= 0) {// Finn plass i siste

// halvdel

p = midten.getNeste();

if (antall != 1) {

Ahoyre++;

}

} else { // Finn plass i f�rste halvdel

p = foerste.getNeste();

Avenstre++;

}

while (el.compareTo(p.getElement()) >= 0) {

p = p.getNeste();

} // while

// Setter inn:

// Innsett foran noden som p peker p�

DobbelNode<T> nyNode = new DobbelNode<T>(el);

// Fyll ut med noen f� setninger

if (antall == 1) {

this.midten = nyNode;

siste.setForrige(nyNode);

foerste.setNeste(nyNode);

nyNode.setNeste(siste);

nyNode.setForrige(foerste);

}else {

nyNode.setForrige(p.getForrige());

p.getForrige().setNeste(nyNode);

nyNode.setNeste(p);

p.setForrige(nyNode);

}

// Oppdaterer ny midten

if (antall % 2 == 1 && antall > 1) { //oppdaterer midten bare når det er odetall antall noder

//nyMidten();

if(Ahoyre > Avenstre) {

midten = midten.getNeste();

} else if (Avenstre > Ahoyre) {

midten = midten.getForrige();

}

Avenstre = antall / 2;

Ahoyre = antall / 2;

}

} // else lovlige

}//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Hjelpemetode til � finne ny midten.

// Mindre effektiv fordi vi m� gjennomg� ca halve listen, men greit nok,

// ellers kan en teste p� om antall er partall er oddetall ved oppdatering

// av midtpeker

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private void nyMidten() {

int midtNR = antall / 2;

DobbelNode<T> p = foerste.getNeste();

for (int i = 1; i <= midtNR; i++) {

p = p.getNeste();

}

midten = p;

}//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*

// \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Override

public boolean fins(T el) {

boolean funnet = false;

DobbelNode<T> p = null;

if ((el.compareTo(foerste.getElement()) <= 0) || (el.compareTo(siste.getElement()) >= 0)) {

// Ugyldig. Alternativt kan vi ha det som et forkrav!

System.out.println("Ugyldig verdi. verdi > " + foerste.getElement() + "verdi < " + siste.getElement());

} else { // Kun lovlige verdier

if (el.compareTo(midten.getElement()) >= 0) { // Let i siste halvdel

p = midten; // Midten defineres � tilh�re siste del

} else { // Let i f�rste halvdel

p = foerste.getNeste();

}

while (el.compareTo(p.getElement()) > 0) {

p = p.getNeste();

} // while

// Test p� funnet

if (el.compareTo(p.getElement()) == 0) {

funnet = true;

}

} // else

return funnet;

}//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*

// \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Omskrive til � bruke finn-metoden

@Override

public T fjern(T el) {

System.out.println("antall hoyre:" + Ahoyre + " antall venstre : " + Avenstre);

T resultat = null;

DobbelNode<T> p = null;

boolean funnet = false;

if ((el.compareTo(foerste.getElement()) <= 0) || (el.compareTo(siste.getElement()) >= 0)) {

// Ugyldig. Alternativt kan vi ha det som et forkrav!

System.out.println("Ugyldig verdi. verdi > " + foerste.getElement() + "verdi < " + siste.getElement());

} else { // Kun lovlige verdier

/\*

if (el.compareTo(midten.getElement()) >= 0) {

p = midten;

} else {

p = foerste.getNeste();

}

while (el.compareTo(p.getElement()) > 0) {

p = p.getNeste();

} // while

\*/

p = finn(el);

if (el.compareTo(p.getElement()) == 0) {

funnet = true;

}

if (funnet) {

// Tar ut

antall = antall - 1;

if (el.compareTo(midten.getElement()) == 0) { //sjekker om man sletter midtelement

if(Ahoyre > Avenstre) {

midten = midten.getForrige();

} else if (Avenstre > Ahoyre) {

midten = midten.getNeste();

} else {

midten = midten.getForrige();

}

}

if (antall % 2 == 1) { //oppdaterer midten bare når det er odetall antall noder

//nyMidten();

//System.out.println("antall hoyre:" + Ahoyre + " antall venstre : " + Avenstre);

if(Ahoyre > Avenstre) {

midten = midten.getForrige();

} else if (Avenstre > Ahoyre) {

midten = midten.getNeste();

}

}

// Fyll ut med noen f� setninger.

p.getNeste().setForrige(p.getForrige());

p.getForrige().setNeste(p.getNeste());

// Oppadtere midten

resultat = p.getElement();

} // funnet

} // lovlige

return resultat;

}//

/\* Alternativ kan fjern-metoden bruke finn-metoden \*/

private DobbelNode<T> finn(T el) {

DobbelNode<T> node = null;

DobbelNode<T> p = null;

// Kun lovlige verdier

if (el.compareTo(midten.getElement()) >= 0) { // Let i siste halvdel

p = midten; // Midten defineres � tilh�re siste del

Ahoyre--;

} else { // Let i f�rste halvdel

p = foerste.getNeste();

Avenstre--;

}

while (el.compareTo(p.getElement()) > 0) {

p = p.getNeste();

} // while

// Test p� funnet

if (el.compareTo(p.getElement()) == 0) {

node = p;

}

return node;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*

// \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public void skrivListe() {

DobbelNode<T> p = foerste;

while (p != null) {

System.out.print(p.getElement() + " ");

p = p.getNeste();

}

System.out.println(

"Foerste:" + foerste.getElement() + " Midten:" + midten.getElement() + " Siste:" + siste.getElement());

System.out.println();

}//

}// class

package no.hvl.dat102.klient;

import no.hvl.dat102.dobbelkjedetordnetlistem.DobbelKjedetOrdnetListeM;

public class KlientDobbelKjedetOrdnetListeM {

public static void main(String[] args) {

String ord[] = { "o", "a", "s", "m", "e", "k", "c" };

DobbelKjedetOrdnetListeM<String> liste = new DobbelKjedetOrdnetListeM(new String("AAA"), new String("zzz"));

// Legger data inn i listen

for (int i = 0; i < ord.length; i++)

liste.leggTil(ord[i]);

// Skriver ut liste

System.out.println("\n\n Opprinnelig liste");

liste.skrivListe();

// Tester om et bestem element fins

String element = new String("Kalle");

boolean funnet = liste.fins(element);

if (funnet)

System.out.println("\nElementet " + element + " fins");

else

System.out.println("\nElementet " + element + " fins ikke");

// Fjerner noen fra listen og skriver ut igjen

String element1 = new String("a");

String element2 = new String("m");

String element3 = new String("e");

String element4 = new String("o");

String element5 = new String("k");

String element6 = new String("c");

String element7 = new String("s");

System.out.println("\n Fjerner " + element1);

liste.fjern(element1);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element2);

liste.fjern(element2);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element3);

liste.fjern(element3);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element4);

liste.fjern(element4);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element5);

liste.fjern(element5);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element6);

liste.fjern(element6);

liste.skrivListe();

System.out.println("\n\n");

System.out.println("\n Fjerner " + element7);

liste.fjern(element7);

liste.skrivListe();

System.out.println("\n\n");

}

}// class

package no.hvl.dat102.dobbelkjedetordnetlistem;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DobbelNode.java

//

// Representer en node i en dobbeltkjedet liste..

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class DobbelNode<T> {

private DobbelNode<T> neste;

private DobbelNode<T> forrige;

private T element;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Oppretter en tom node node.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public DobbelNode() {

neste = null;

element = null;

forrige = null;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Oppretter en node som lagrer det spesifiserte elementet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public DobbelNode(T elem) {

neste = null;

forrige = null;

element = elem;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Returnerer en referanse til etterf�lgeren av denne noden, eller null hvis

\* denne er siste.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public DobbelNode<T> getNeste() {

return neste;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Returnerer forgjengeren til denne noden, eller null hvis denne er f�rste

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public DobbelNode<T> getForrige() {

return forrige;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Setter neste til � peke p� dnode.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void setNeste(DobbelNode<T> dnode) {

neste = dnode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Settter ny forrige til � peke p� dnode.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void setForrige(DobbelNode<T> dnode) {

forrige = dnode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Returnerer elementet til denne noden..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public T getElement() {

return element;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lagrer nytt element i denne noden..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void setElement(T elem) {

element = elem;

}

}

c) Oppdaterer midten ved å bruke getForrige() eller getNeste() metodene på midten når man legger til et nytt objekt, og må bestemme om hvilke metode man skal bruke ut ifra hvordan listen ser ut.

d)

i) gjennomsnitt : n/2 + 0.5: tar alle tilfellene fra 1 sammenliging (finner det på første element), fram til n elementer og deler på antall elementer.

verste tilfellet: n : finner det på siste element (går gjennom alle elementer)

ii) gjennomsnitt : n/4 + 0.75 leter bare gjennom halve tabellen, så blir som i «i)»

verste tilfellet: n/2 blir som i «i» bare med at du søker gjennom halve tabellen.

iii) gjennomsnitt: ca n/8 siden man da halverer samlingen enda en gang

verste tilfellet: n/4 siden den er halvert enda en gang.

# Oppgave 3

b)

Ønsker å søke etter nummer 8 I tabellen og retunere plassen i tabellen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 | 15 | 18 | 21 | 23 | 27 | 29 | 30 | 31 |

1. Først setter man start = 0 og slutt = tab.length – 1
2. så finner man midten og sjekker om tallet er lik tallet i denne posisjonen
3. om ikke må man sjekke om tallet er større eller mindre en midt pekeren, er den større kutter vi ut alt til venstre + midt pekeren og gjøre et nytt søk. Er tallet vi leitere etter mindre en midt pekeren kutter vi ut alt til høgre + midt pekeren og gjøre et nytt søk på venstre delen.

i dette tilfelle vil neste tabell se slik ut

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 |

Prosessen igjen tar seg og vi får i neste runde denne tabellen siden 8 er større en 7 går vi til høgresiden i tabellen

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 6 |
| 8 | 10 | 12 |

Samme skjer igjen og vi går til veneste siden 8 er mindre en 10. vi får da at tall == tab[mid] og vi returnerer posisjonen vi fikk match i

c)

Ønsker å søke etter nummer 16 I tabellen og returnere plassen i tabellen eller -1 om den ikke finnes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 | 15 | 18 | 21 | 23 | 27 | 29 | 30 | 31 |

Finner at 16 er størren en midt pekeren (15). hele venstresiden og midt pekeren blir kuttet ut.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 18 | 21 | 23 | 27 | 29 | 30 | 31 |

Finner at midt pekeren ikke er like tallet vi leter etter men mindre så kutter ut høgresiden pluss midt pekeren

|  |  |  |
| --- | --- | --- |
| 8 | 9 | 10 |
| 18 | 21 | 23 |

Finner at midt pekeren ikke er like tallet vi leter etter men mindre så kutter ut høgresiden pluss midt pekeren

|  |
| --- |
| 8 |
| 18 |

Forsatt er 16 midre en 18 så man kommer ut av løkka her og retunerer -1:

While (start<=slutt)

|  |  |  |
| --- | --- | --- |
| 1 | Start = 8 | Slutt = 14 |
| 2 | Start = 8 | Slutt = 10 |
| 3 | Start = 8 | Slutt = 8 |
| 4 | Start = 8 | Slutt = 7 |

Vi får 1 nøkkelsammenligning pr. rekursiv kall. I verste tilfelle blir det log2n. Hvordan stemmer det med søk av 16 som ikke fins? Det stemmer.

# Oppgave 4

package no.hvl.dat102;

import java.util.Arrays;

import java.util.Random;

import java.util.Scanner;

public class SorteringTest {

public static Integer[] listeGenerator(int størrelse) {

Integer[] Tabell = new Integer[størrelse];

Random randTall = new Random();

for(int i = 0; i < størrelse; i++) {

Tabell[i] = randTall.nextInt(100);

}

return Tabell;

}

public static void main(String[] args) {

long totaltid = 0;

long Starttid = System.nanoTime();

for(int i = 0; i < 100; i++) {

Integer[] tabell = listeGenerator(128000);

// System.out.println("Listen før fletting" + "\n" + Arrays.toString(tabell));

// flettesortering test

Flettesortering.mergeSort(tabell);

// boblesortering test

//Boblesortering.bubbleSort(tabell);

// QuickSort Test

//QuickSort.quickSort(tabell);

// sortering ved innsetting Test

//SorteringVedInnsetting.innsettingSort(tabell);

// sortering ved utvalg

// SorteringVedUtvalg.selectionSort(tabell);

}

long sluttid = System.nanoTime();

System.out.println("gjennomsintlig kjøretid = " + (sluttid - Starttid) / 100 + " ns");

// System.out.println("Listen etter fletting" + "\n" + Arrays.toString(tabell));

// System.out.println("Kjøringen tok " + (sluttid - Starttid) + " ns");

}

}

package no.hvl.dat102;

public class SorteringVedInnsetting {

public static <T extends Comparable<T>> void innsettingSort(T[] tab) {

for (int index = 1; index < tab.length; index++) {

T Nøkkel = tab[index];

int posisjon = index;

while (posisjon > 0 && tab[posisjon - 1].compareTo(Nøkkel) > 0) {

tab[posisjon] = tab[posisjon - 1];

posisjon--;

}

tab[posisjon] = Nøkkel;

}

}

}

package no.hvl.dat102;

public class SorteringVedUtvalg {

public static <T extends Comparable<T>> void selectionSort(T[] list) {

for (int i = 0; i < list.length - 1; i++) {

int iSmallest = i;

for (int j = i + 1; j < list.length; j++) {

if (list[iSmallest].compareTo((list[j])) > 0) {

iSmallest = j;

}

}

T iSwap = list[iSmallest];

list[iSmallest] = list[i];

list[i] = iSwap;

}

}

}

package no.hvl.dat102;

import java.util.ArrayList;

import java.util.Arrays;

public class RadixSortering {

public static void main(String[] args) {

long startTid = System.nanoTime();

for(int j = 0; j<10; j++) {

Integer [] tabell = SorteringTest.listeGenerator(128000);

final int RADIX = 10;

ArrayList<Integer> bøtteTabell[] = new ArrayList[RADIX];

for (int teller = 0; teller < bøtteTabell.length; teller++) {

bøtteTabell[teller] = new ArrayList<>();

}

// sorter inn i bøtetabell

boolean maxTallnåd = false;

int temp = -1, posisjon = 1;

while (!maxTallnåd) {

maxTallnåd = true;

for (Integer element : tabell) {

temp = element / posisjon;

bøtteTabell[temp % RADIX].add(element);

if (maxTallnåd && temp > 0) {

maxTallnåd = false;

}

}

// plasserer tilbake i oprinnelig tabell

int a = 0;

for (int b = 0; b < RADIX; b++) {

for (Integer i : bøtteTabell[b]) {

tabell[a++] = i;

}

bøtteTabell[b].clear();

}

posisjon = posisjon \* RADIX;

}

//System.out.println("Listen etter fletting" + "\n" + Arrays.toString(tabell));

}

long slutTid = System.nanoTime();

System.out.println("Gjennomsnitts tid " + (slutTid - startTid) / 10 + " ns" );

}

}

package no.hvl.dat102;

public class Boblesortering {

private static <T extends Comparable<T>> void bytterOm(T[] tab, int min, int max) {

T temp = tab[min];

tab[min] = tab[max];

tab[max] = temp;

}

public static <T extends Comparable<T>> void bubbleSort(T[] tab) {

int posisjon, scan;

for (posisjon = tab.length - 1; posisjon >= 0; posisjon--) {

for (scan = 0; scan <= posisjon - 1; scan++) {

if (tab[scan].compareTo(tab[scan + 1]) > 0) {

bytterOm(tab, scan, scan + 1);

}

}

}

}

}

package no.hvl.dat102;

public class Flettesortering {

@SuppressWarnings("ikke skjekket")

public static <T extends Comparable<T>> void mergeSort(T[] list) {

if (list.length > 1) {

T[] firsthalf = (T[]) new Comparable[list.length / 2];

System.arraycopy(list, 0, firsthalf, 0, list.length / 2);

mergeSort(firsthalf);

// sikkerer at vi får med oss alt i tilfelle list er av oddetall

int secondHalfLength = list.length - (list.length / 2);

T[] secondHalf = (T[]) new Comparable[secondHalfLength];

System.arraycopy(list, list.length / 2, secondHalf, 0, secondHalfLength);

mergeSort(secondHalf);

T[] temp = merge(firsthalf, secondHalf, list);

System.arraycopy(temp, 0, list, 0, temp.length);

}

}

public static <T extends Comparable<T>> T[] merge(T[] firsthalf, T[] secondHalf, T[] list) {

int firstHalfIndex = 0, secondHalfIndex = 0, listIndex = 0;

while (firstHalfIndex < firsthalf.length && secondHalfIndex < secondHalf.length) {

if (firsthalf[firstHalfIndex].compareTo(secondHalf[secondHalfIndex]) < 0) {

list[listIndex++] = firsthalf[firstHalfIndex++];

} else {

list[listIndex++] = secondHalf[secondHalfIndex++];

}

}

while (firstHalfIndex < firsthalf.length) {

list[listIndex++] = firsthalf[firstHalfIndex++];

}

while (secondHalfIndex < secondHalf.length) {

list[listIndex++] = secondHalf[secondHalfIndex++];

}

return list;

}

}

package no.hvl.dat102;

public class QuickSort {

public static <T extends Comparable<T>> void quickSort(T[] tab) {

quickSort(tab, 0, tab.length - 1);

}

private static <T extends Comparable<T>> void bytterOm(T[] tab, int min, int max) {

T temp = tab[min];

tab[min] = tab[max];

tab[max] = temp;

}

private static <T extends Comparable<T>> int partition(T[] tab, int min, int max) {

T partitionElement;

int left, right;

int mid = (min + max) / 2;

partitionElement = tab[mid];

bytterOm(tab, mid, min);

left = min;

right = max;

while (left < right) {

while (left < right && tab[left].compareTo(partitionElement) <= 0) {

left++;

}

while (tab[right].compareTo(partitionElement) > 0) {

right--;

}

if (left < right) {

bytterOm(tab, left, right);

}

}

bytterOm(tab, min, right);

return right;

}

private static <T extends Comparable<T>> void quickSort(T[] tab, int min, int max) {

if (min < max) {

int indexofpartion = partition(tab, min, max);

quickSort(tab, min, indexofpartion - 1);

quickSort(tab, indexofpartion + 1, max);

}

}

}

a & b)

Finne c:

Ink Drawings
Ink Drawings
￼
￼
￼￼￼
￼
￼￼￼
￼


Flettesortering:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32000 | 10 | 30 714 830 ns | 368756924.8 ns |
| 64000 | 10 | 106 019 569 ns | 78679384.5 ns |
| 128 000 | 10 | 39505632 ns | 1672147699 ns |

QuickSort:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32000 | 10 | 29 056 123 ns | 368756924.8 ns |
| 64000 | 10 | 63 992 861 ns | 786793849.5 ns |
| 128 000 | 10 | 108543528 ns | 1672147699 ns |

Bublesort:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 10 463 630 ns | 770000000 ns |
| 5000 | 10 | 140 497 025 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 520 784 287 ns | 7.7\*10^10 ns |

Sortering ved utvalg: O(n^2)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 10 393 641 ns | 770000000 ns |
| 5000 | 10 | 62 620 307 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 334 246 769 ns | 7.7\*10^10 ns |

Sortering ved Insetting: O(n^2)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 16 090 953 ns | 770000000 ns |
| 5000 | 10 | 39 079 507 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 143 246 810 ns | 7.7\*10^10 ns |

Sortering ved Radix sort: O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32 000 | 10 | 5 812 596 ns | 24 640 000 ns |
| 64 000 | 10 | 9 337 011 ns | 49 280 000 ns |
| 128 000 | 10 | 13 305 899 ns | 98 560 000 ns |

I følge testene våre er quicksort er raskest på mindre elementer, men flettesort blir raskere jo flere elementer det er i tabellen.

c)

Ved å bruke sirkulær kø slipper man å flytte elementene når man ‘popper’

Usortert tabell.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 123 | 398 | 210 | 019 | 528 | 513 | 129 | 294 |

Sorterer inn i midlertidige tabeller med tallet lengst til høgre som utgangspunkt. Starter med tallet først i tabeller (tab[0]).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 210, |  |  | 123,513 | 294 |  |  |  | 398,528 | 019,129 |

Setter så tilbake i opprinnelige tabell i den rekkefølgen de ble satt inn i de midlertidige tabellene og kjører sortering med hensyn på det midterste tallet

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 210 | 123 | 513 | 294 | 398 | 528 | 019 | 129 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 210,513,019 | 123,129 | 528 |  |  |  |  |  | 294,398 |

Setter tilbake i tabellen og sorterer igjen med hensyn på siste tallet

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 210 | 513 | 019 | 123 | 129 | 528 | 294 | 398 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 019 | 123,129 | 210,294 | 398 |  | 513,528 |  |  |  |  |

Setter tilbake i tabellen og ser at tabellen er sortert

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 019 | 123 | 129 | 210 | 294 | 398 | 513 | 528 |

Vi får data sortert til slutt ved å ‘poppe’ alle køene i stigende rekkefølge og vil da få ut tallene sortert.

d)

Når n = 100:

vanlig kvikksort: 483938 ns

Nye kvikksort: 630276 ns

Da vil det si at den nye metoden er raskere en vanlig kvikksort