Eksamen

1a)

Viser en splitt ved å skrive bindestrek. Tallene står sammen når de er komma etter hverandre.

2, 9, 5, 4, 6, 1, 4, 3

Splitt: 2, 9, 5, 4 - 6, 1, 4 , 3

Splitt2: 2, 9 - 5, 4 – 6, 1 – 4, 3

Splitt3: 2 – 9 - 5 – 4 - 6 - 1 - 4 - 3

Merge: 2,9 – 4,5 – 1,6 - 3,4

Merge2: 2, 4, 5, 9 – 1, 3, 4, 6

Merge2: 1, 2, 3, 4, 5, 6, 9 **FERDIG!**

1b)

Forslag1. Du kan gjøre delsortering ved at man benytter for-loopene og passer på at starten og slutten begynner og slutter innenfor intervallet. Da hopper man over alle indexer som er utenfor intervallet.

I forhold til sann jeg tenker er dette mulig på alle tre sorteringsmetodene.

Forslag 2: Ett annet forslag som jeg ikke rakk å programmere på 1c som kanskje er mer riktig: Gå igjennom hele tabellen i den første loopen og bare sammenligne alle tallene som ligger inni intervallet i den andre loopen. Dersom den har plass puttes den inn. Tall byttes ut og flyttes inn i intervallet etter hvert som man går igjennom tabellen. Når dette er ferdig skal tallene ligge i intervallet som om de er sortert i forhold til hele tabellen. Men det er bare tallene inni som stemmer.

1c)

Insertion sort med intervall.

```
public static<T extends Comparable<? Super T >> void delSorter(T[] a, int low, int high){
        for (int j = low; j < high; j++)
        {
                T nøkkel = a[j];
                int i = j-1;
                while ( (i > -1) && ( a[i].compareTo(nøkkel) > 0 ) )
                {
                        Swap(a[i+1], a[i]);
```

```
                    i--;
                }
            a[i+1] = nøkkel;
            }
}
```

1d)

```
public static <T extends Comparable<? Super T>> boolean delSortert(T[] a, int low, int high){
        for(int i=low, i<high; i++){
                if(a[i].compareTo(a[i+1]) > 0)
                        return false; //siden den nå måtte gjøre en swap om den var usortert
        }
        return true; //Har kommet seg gjennom hele tabellen uten feil.
}
// algoritmens orden vil være O(n) siden den går igjennom loopen n ganger
```

Oppgave 2 – oppgave a,b,c,d,e,f

```
Public class Vassdrag {
        protected Elvestrekning rot;
        protected String navn;
        protected int størrelse = 0;

         public Vassdrag(string navn) {
                this.navn = navn;
         }
        public void setInn(String elveInfo) throws Exception {
                string[] splitTab = elveinfo.split(« »); //Splitter på mellomrom og putter inn i tabell.
                If(splitTab.length <6)
                        Throw new Exception(«Ikke nok verdier i filen»);
                ElveStrekning elv = new Elvestrekning(splitTab[6], splitTab[5], splitTab[2], splitTab[3],
                splitTab[4]);
```

```
String [] plassTab = splitTab[0].split(«.»); //Splitt på punktum.
int pathIndeks = 1;
ElveStrekning forelder = rot;
ElveStrekning nåElv = null;

If(plassTab.length== 1)
        rot =elv;
        return;
else{
        while(pathIndeks < plassTab.length){
                String path = plassTab[pathIndeks];

                if(path.equals(«1»)){
                        if(forelder.v == null){
                                nåElv.v = elv;
                                return;
                        }
                        If(forelder.v == null && pathIndeks < plassTab.length-1)
                                Throw new Exception(«Retning i fil eksisterer ikke»);

                        nåElv = forelder.v;
                }else if(path.equals(«2»)) {
                        if(forelder.h == null){
                                nåElv.h = elv;
                                return;
                        }
                        If(forelder.h == null && pathIndeks < plassTab.length-1)
                                Throw new Exception(«Oppført feil retning i fil»);

                        nåElv = forelder.h;
                }
```

```
                pathIndeks++;
            }
            this.størrelse++;
        }

public void lesInn(File fil){
        try{
            FileReader filskriver = new FileReader(fil);
            Scanner vassScanner = new Scanner(filskriver);

            while(vassScanner.hasNextLine()){
                settInn(vassScanner.nextLine());
            }
            vassScanner.close();
        }catch(Exception e) { e.printStackTrace() }
        }
}
public boolean sjekkStruktur() {
        Kø<ElveStrekning> kø = new Kø();
        Kø.insert(rot);
        int størrelsen = 0;
        While(!kø.isEmpty()){
            ElveStrekning elv = kø.remove();
            if((elv.v == null && elv.h == null) ||  (elv.v != null && elv.h != null)){
                if(elv.v != null) kø.insert(elv.v);
                if(elv.h != null) kø.insert(elv.h);
                størrelsen++;
            }
            else break;
        }
        If(størrelsen == this.størrelse)
```

```
        return false; //Vi gikk ikke gjennom hele treet før den ble avbrytt.


        return true;
}

public double vannUt(){
        return rot.vannUt();
}

public float areal(){
        Kø<ElveStreking> kø = new Kø();
        Kø.insert(rot);
        float nedbørAreal = 0.0f;
        While(!kø.isEmpty()){
                ElveStrekning elv = kø.remove();
                nedBørAreal = elv.nedbør * elv.lengde;  //Usikker på selve regnestykket.
                if(elv.v == null && elv.h == null){
                        if(elv.v != null) kø.insert(elv.v);
                        if(elv.h != null) kø.insert(elv.h);
        }
        return nedbørAreal;

}

public float lengde(ElveStrekning elv, int lengde ) {
        if(elv == null) return 0.0f;

        int venstreElv = lengde(elv.v, lengde);
        int venstreHøyde = lengde(elv.h, lengde)
        lengde += elv.lengde;
        lengde += Math.max(venstreElv, høyreElv);
```

```
            return lengde;
        }


}
Public class Elvestrekning {
        String navn; // Navn på elvestrekning
        float nedbør; // forventet nedbør i mm i området
         float lengde; // lengden på elvestrekningen i km
        float tilsigAreal; // tilsigareal i km2
        float fallHøyde; // høydeforskjell mellom øvre og nedre ende, i meter
        Elvestrekning v, h; // venstre og høyre barn i treet
         public Elvestrekning(String navn, float nedbør, float lengde, float tilsigAreal, float fallHøyde) {
            this.navn = navn;
            this.nedbør = nedbør;
            this.lengde = lengde;
            this.tilsigAreal = tilsigAreal;
            this.fallHøyde = fallHøyde;
         }

        double tilsig() { return nedbør*tilsigAreal*1000; }  //bekker osv

        //vann frå andre elver. Er aldri null. Rekursiv ikke stopp her
        private double vannInn(double vann, ElveStrekning elv) {
            int totalVann = vann + tilsig();
            return vannUt(totalVann,this.v) + vannUt(totalVann, this.h);
        }

        //Drivermetode
        Public vannUt(){
```

```
            return vannUt(0, this);
        }

        //Summen av tilsig og vannInn. Rekursiv stoppes her.
        public double vannUt(double vann, ElveStrekning elv) {
            if(this.v == null && this.h == null)
                return 0;

            int vannUtSumV = vannInn(vann, elv.v);
            int vannUtSumH = vannInn(vann, elv.h);

            return vannUtSumV + vannUtSumH;
        }
}
```

2g)

Selve tekstfilen kan innehold n elementer. Vi går gjennom denne i while n antall ganger i les inn metoden.

Dermed så settes elementet inn i treet i settInn metoden, ved å halvere resultatet nedover til det finner riktig plass. Dermed går vi gjennom treet logn ganger.

Derfor svarer jeg: Average: O(nLogn)

Best: O(1) – Om det bare finnes en linje i tekstfilen og den henviser seg til rotnoden. For da går den ikke inn i while i settInn og går bare en runde i while

I lesInn.