Løsningsforslag 2017 eksamen

Oppgave 1: O-notasjon (maks 8 poeng)

1.
   (i) O(n) gir 2 poeng, O(100n) gir 1 poeng
   (ii) O(n^2) gir 1 poeng
   (iii) O(n log n) gir 2 poeng
2. (i) er mest effektiv i henhold til O-notasjon. 1 poeng.
3. Ja. For små n er både (ii) og (iii) mer effektiv enn (iii). Hadde innmaten i løkken vært lik, ville vi forventet at (iii) er mer effektiv for n < 2^100, dvs så lenge log n er mindre enn 100. 2 poeng for riktig svar med noenlunde begrunnelse. 0 poeng for «Ja» uten begrunnelse.

Oppgave 2: Binære søketrær

2a (maks 5 poeng)

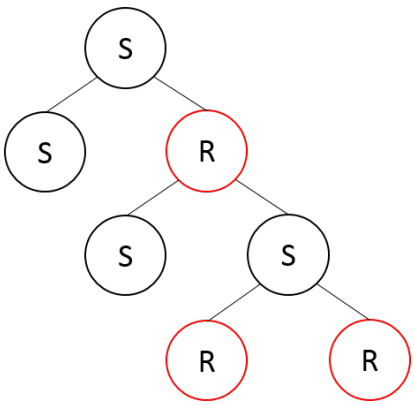
```
boolean sjekkStruktur(BinNode r, BinNode s) {
    if (r == null && s == null) return true;
    if (r != null && s != null) {
        return (sjekkStruktur(r.venstre, s.venstre) &&
            sjekkStruktur(r.hoyre, s.hoyre);
    }
    return false;
}
```

Hovedpoenget med denne oppgaven er å vise at man behersker traversering av trær. Det er ikke noe krav om at oppgaven løses rekursivt, selv om det helt klart er enklest. Trekk max 2 poeng for feil/manglende håndtering av null. Det står tydelig i oppgaveteksten at man skal se bort fra verdiene (elementene) i nodene, så det trekkes 1 poeng hvis det likevel gjøres noen sammenligninger her.

2b (maks 5 poeng)

```
boolean sjekkAVL(BinNode n) {
    // Bruker -2 som feilverdi i hjelpemetoden.
    return (sjekkHoyde(n) != -2);
}

int sjekkHoyde(BinNode n) {
    if (n == null) return -1;

    int v = sjekkHoyde(n.venstre);
    if (v == -2) return v;

    int h = sjekkHoyde(n.hoyre);
    if (h == -2) return h;

    if (Math.abs(v-h) > 1) return -2;
    return (Math.max(v,h) + 1);
}
```

2c (maks 4 poeng)

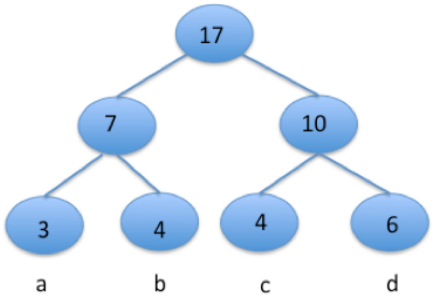


2 poeng for et korrekt rød-svart tre, 2 poeng dersom det i tillegg IKKE er et AVL-tre. Kan gi totalt 2 poeng dersom det bare er 1-2 små feil i det rød-svarte treet, men studenten viser å ha forstått hovedpoenget.

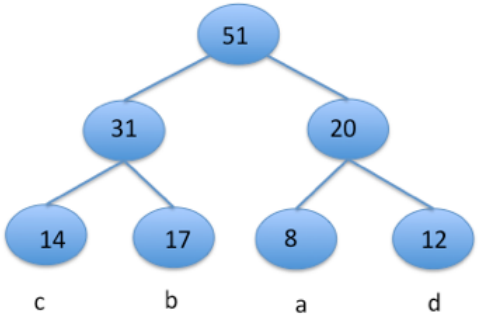
Oppgave 3: Huffman

3a (maks 8 poeng)

A er ikke fordi node med vekt 17 har bare et barn og vektene er feil.



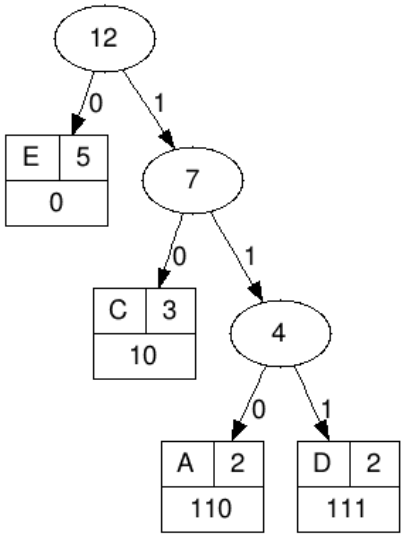
B er ikke fordi nodene med vekter 8 og 12 burde være kombinert først.



- 4 poeng for hvert tre. 2 poeng for riktig forklaring. 2 poeng for riktig tre.

  Ingen poeng for endring av vektene på alfabetet.

3b (maks 5 poeng)



Oppgave 4: CIA operasjon

4a (maks 4 poeng)

Use depth first search on graphs. Nodes are Spies and edges are channels. It is an undirected graph. The edges are marked/weighted as 'secure' or 'not secure'. The problem is equivalent to finding the number of connected components where a component is connected in the sense of only having edges marked as 'secure'.

4b (maks 9 poeng)

```
class Channel{
        Spy a;
        Spy b;
        boolean secure; // true if this is a secure channel between a and b, false
otherwise

     Channel(Spy a, Spy b, boolean secure) {
        this.a = a;
        this.b = b;
        this.secure = secure;
     }


    Spy destination(Spy source) {
        return (source == a) ? b : (source == b) ? a : null;
    }
}


class Spy{
 List <Channel> channels;
 Boolean visited = false;
.....

}


int numberOfSpies {List <Spy>  spies }{
  int counter = 0;
  for (Spy spy:spies) {
        if (spy.visited == false){
        spy.visited = true;
        counter ++;
        dfs(spy); }
  }
 return counter;
}
```

```
void dfs(Spy spy){
    for (Channel channel: spy.channels){
          spy candidate = channel.desitination(spy);
          if (!candidate.visited && channel.secure){
             candidate.visited = true;
             dfs (candidate);
          }
   }
}
```

4c (maks 5 poeng)

Assign the secure edges weight 0 and the unsecure edges weight 1 and run an algorithm to compute MST. The MST will be a spanning tree, and minimizing the total weight is equivalent to minimizing the heavier unsecure channels. The sum of the MST corresponds to the number of unsecure channels.
3 poeng for riktig svar her.

Note that the text says ”utvalgte andre spioner”/ ”selected spies” (also the example graph given to the student is not connected), so the graph can be disconnected. The most correct answer is to use Kruskal algorithm to compute a minimal spanning forest.
2 poeng for Kruskal.

If Prim is used then one must remember to start Prim selveral times, once for each connected component, to cover all nodes.
1 poeng for Prim.

4d (maks 4 poeng)

Use Kruskal. The CIA operator must use minimal 3 unsecure channels either
(G, H, false), (D, F, false), and (C, E, false)
Or
(C, E, false), (B, C, false), and (G, H, false)
2 poeng for riktig identifisering.

Removing one unsecure channel from a MST will result in a disconnected component, hence will require one more direct communication to a spy. Hence the unsecure channel is neccessary inorder to minimize number of direct communications. There is no need to prove that Kruskal or Prim computes MST.
2 poeng for riktig svar her.

Oppgave 5 (maks 15 poeng)

Den naturlige løsningen er å ta utgangspunkt i programmet fra kompendiet/forelesning. I stedet for en boolsk brukt-array, kan det brukes en int-array over hvor mange ganger det fortsatt er mulig å bruke et siffer, initialisert til 2. I stedet for å teste !brukt[siffer] blir testen da brukt[siffer] > 0. I tillegg må det testes at det ikke kommer to like på rad.

```
class Gen2 {
    int n;
    int[] array = new int[2*n];
    int[] brukt = new int[n];
    int antall = 0;

    public static void main(String[] args) {
     Gen2 g = new Gen2();
     g.init(Integer.parseInt(args[0]);
     g.gen(0);
    }
    public void init(int antall) {
     n = antall;
     for (int i = 0; i < n; i++) {
         brukt[i] = 2;
     }
    }
    public void gen(int i) {
     if (i < 2*n) {
         for (int tall = 0; tall < n; tall++) {
          if (brukt[tall]>0) {
              if (i == 0 || array[i-1] != tall) {
               brukt[tall]--;
               array[i] = tall;
               gen(i+1);
               brukt[tall]++;
              }
          }
         }
     } else {
         for (int j = 0; j < array.length; j++) {
               System.out.print(array[j] + " ");
         }
         System.out.println(" ");
     }
    }
}
```

Det viktigste her er sekvensgenereringen, men studentene skal også vise at de kan lage dette til et komplett program. Ren kopi av forelesening/kompendium uten reelle forsøk på tilpasninger gis 0 poeng. Bruk ellers skjønn, med utgangspunkt i følgende poengfordeling:

Hvert siffer kan komme max to ganger – 7 poeng.
Avskjæring slik at to like sifre ikke kommer rett etter hverandre – 4 poeng.
Komplett program med main, initialisering osv – 4 poeng.
En fullgod alternativ løsning vil være å tilpasse den andre permutasjonsteknikken som står skrevet i kompendiet, men den er ikke forelest så vi forventer ikke at så mange studenter vil bruke den. Andre løsninger aksepteres også, men trekk for unødvendig komplisert kode eller ineffektive løsninger (for eksempel å lete gjennom hele arrayen for å finne ut om man kan bruke et siffer eller ikke).

Oppgave 6: Sortering

6a (maks 13 poeng)
Viktige poeng for løsningen

- **a)** Vi må også sortere over til b[] lengder == 1 (dvs**.** >= tegn: **if (right-left >= 0)** {
  Trekk på 3punkter for ikke å se det
- **b)** Vi leser fra a[] (**t = a[k];)** og kopierer denne inn i b[])
  Viktig bare å lese a[] – uten det er oppgaven ikke løst
- c) Første element i a[] kopieres over i b[] (**b[left] = a[left];**
  Trekk 2 punkter for ikke å gjøre det

```
// sort a[left..right] over to b[left..right]. b might be a
   void insertABSort(int a[],int [] b,int left, int right){
     if (right-left >= 0) {
        int i, t;

        b[left] = a[left];
        for (int k = left+1 ; k <= right ; k++){
            t = a[k];
            i = k-1;
           while (i >= left && b[i] > t)
              { b[i+1] = b[i]; i--;}
           b[i+1] = t;
       } // end for k
     } // end left <= right
} // end insertABSort
```

6b (maks 5 poeng)

Å bruke a på begge parametereplassene går BRA fordi vi leser bare fra første parameterplass og skriver I andre parametereplass – nå a.
Best ser vi at det går greit hvis vi setter inn a for b i løsningen på Deloppgave 1. Vi ser at koden bare da gjør bare en overflødig instrusjon (a[left] = a[left];) – ellers blir dette vanlig innstikk-sortering.

Oppgave 7: Søking

7a (maks 6 poeng)
Husk ingen poeng for å sortre først. Begge algoritmene er O(n*n) , (men bruker vi innstikksortering først er den jo også O(n*n)).

```
static double median1(int [] a) {
        int medianLen = a.length/2;  // antar a.length er et oddetall
        int val=0, numMindre;
            for(int i =0; i < a.length; i++) {
              numMindre =0;
              val = a[i];
              for (int j =0; j < a.length; j++) {
                 if (a[j] < val) numMindre= numMindre+1;
              }
              if (numMindre == medianLen) break;
            } // end i
            return val;
          }// end median1
```

7a (maks 4 poeng)

```
int  median2(int [] a) {
    int medianLen = a.length/2 +1; // antar a.length er et oddetall
        int  numMindre, numStorre;;
        int val=0;
        for(int i =0; i < a.length; i++) {
            numMindre =0; numStorre =0;
            val = a[i];
            for (int j =0; j < a.length; j++) {
                if (a[j] <= val) numMindre++;
                if (a[j] >= val ) numStorre++;
            }
            if ((numMindre >=  medianLen)
                        && (numStorre >= medianLen)) break;
        } // end i
    return val;
}
```