Algoritmer, datastrukturer och komplexitet, hösten 2016
Lösningsförslag till mästarprov 1: algoritmer

1. **Uppgiftsutdelning**
   Bygg enligt ledningen en graf som har kanter mellan de hörn som representerar studenter som känner varandra.

   Problemet kan lösas genom att man försöker färga grafen med två färger, vitt för de hörn som motsvarar studenter som ska lösa den ena uppgiften och svart för de övriga. Om det går bra att tvåfärga grafen så räcker det med två uppgifter, men om det inte går att tvåfärga grafen så behöver läraren göra fler än två uppgifter. Detta är alltså en reduktion av uppgiftsutdelningsproblemet till problemet tvåfärgning av graf.

   När man valt färg på ett första hörn i en sammanhängande komponent så är det unikt bestämt vilken färg övriga hörn måste ha; alla grannar till ett vitt hörn måste ju vara svarta. Vi kan därför lösa problemet genom att gå igenom varje sammanhängande komponent med en djupetförstsökning där vi färgar grannarna enligt följande regler:

   - Om grannen inte har någon färg så färgar vi den med motsatt färg och gör sedan en djupetförstsökning med start i detta hörn.
   - Om grannen redan har motsatt färg så gör vi ingenting.
   - Om grannen redan har samma färg så är det omöjligt att färga den här komponenten med två färger och vi avbryter algoritmen.

   I pseudokod:

```
checkassignments(pairs)=
  # Skapa en graf
  låt G=(V,E) där V och E initialt är tomma
  för varje par (s, t) i pairs:
     om s inte finns i V: lägg till s till V
     om t inte finns i V: lägg till t till V
     lägg till (s,t) till E

  # Besök varje sammanhängande komponent
  för varje hörn v i grafen G:
     om v ännu inte har någon färg:
       färga v vit
       DFS(G, v)
  return 'stämmer'

  # Försök tvåfärga en sammanhängande komponent i G
  # med start i (det redan färgade) hörnet v.
  DFS(G, v):
    för alla grannar w till v:
      om w inte har någon färg:
        färga w med motsatt färg till v
        DFS(G, w)
      om w har samma färg som v:
        return 'stämmer inte' # avbryt exekveringen
```

   Algoritmen gör ett konstant arbete för varje hörn och varje kant i grafen så tidskomplexiteten blir $O(|V| + |E|)$, där $|V|$ är antalet hörn och $|E|$ antalet kanter. I det ursprungliga problemet blir tidskomplexiteten $O(n + m) = O(m)$ där n är antalet studenter och m är antalet vänskapsrelationer. Analysen förutsätter att grafen lagras med grannlistor och att V kan indexeras med studentnummer.

2. **Optimal Pokémonvandring med jaktfri ruvning**
För att lösa problemet använder vi dynamisk programmering. Låt delproblemet $S[a_2, a_5, a_{10}]$ vara maximala antalet pokéstopp som Arrietty kan nå under dom första $2a_2+5a_5+10a_{10}$ km av vandringen. När vi beräknat alla $S[a_2, a_5, a_{10}]$ hittar vi det sökta svaret i $S[n_2, n_5, n_{10}]$.

Basfallet är trivialt: $S[0, 0, 0] = 0$. Rekursionssteget för $S[a_2, a_5, a_{10}]$ är enkelt att härleda. Det sista ägget som Arrietty använde i sin maximala $2a_2 + 5a_5 + 10a_{10}$-kilometersvandring var antingen ett 2-kilometersägg, ett 5-kilometersägg eller ett 10-kilometersägg. När hon började vandra med det ägget hade hon maximalt nått $S[a_2 - 1, a_5, a_{10}]$, $S[a_2, a_5 - 1, a_{10}]$ respektive $S[a_2, a_5, a_{10} - 1]$ pokéstopp (beroende på vilket ägg det var). Det största av dessa tre värden måste vara det bästa (ägget) att välja. Och när det ägget kläcks kommer hon att nå ytterligare $P[2a_2 + 5a_5 + 10a_{10}]$ pokéstopp.

Rekursionen kan därför skrivas
$S[0, 0, 0] = 0$
$S[a_2, a_5, a_{10}] = \max(S[a_2-1, a_5, a_{10}], S[a_2, a_5-1, a_{10}], S[a_2, a_5, a_{10}-1])+P[2a_2+5a_5+10a_{10}]$

Bara termerna i max-uttrycket i rekursionen som har ickenegativa index ska tas med.

Vi beräknar värdena i den tredimensionella matrisen S för stigande värden i en dimension i taget, t ex för stigande a_{10} för stigande a_5 för stigande a_2. Då finns alltid dom värden vi behöver använda i rekursionen redan beräknade.

Så här blir algoritmen:

$S[0, 0, 0] \leftarrow 0$
assertion Basfallet beräknat enligt rekursionen
for $a_2 \leftarrow 0$ **to** n_2 **do**
  for $a_5 \leftarrow 0$ **to** n_5 **do**
    for $a_{10} \leftarrow 0$ **to** n_{10} **do**
      $t \leftarrow 0$
      if $a_2 > 0$ **then** $t \leftarrow S[a_2 - 1, a_5, a_{10}]$
      if $a_5 > 0$ **and** $S[a_2, a_5 - 1, a_{10}] > t$ **then** $t \leftarrow S[a_2, a_5 - 1, a_{10}]$
      if $a_{10} > 0$ **and** $S[a_2, a_5, a_{10} - 1] > t$ **then** $t \leftarrow S[a_2, a_5, a_{10} - 1]$
      $S[a_2, a_5, a_{10}] \leftarrow t + P[2a_2 + 5a_5 + 10a_{10}]$
      assertion $S[a_2, a_5, a_{10}]$ beräknat enligt rekursionen
return $S[n_2, n_5, n_{10}]$

Tidskomplexiteten domineras av dom tre nästlade slingorna som går n_2, n_5 respektive n_{10} varv. Arbetet inuti den innersta slingan är konstant. Totalt blir det $O(n_2 n_5 n_{10})$.

Rekursionens korrekthet och beräkningsordningen motiveras ovan och det är enkelt att se att beräkningen av basfall och rekursionssteg sker korrekt enligt assertion-satserna i pseudokoden.

3. **Boka klassrum optimalt**
Problemet är en generalisering av intervallschemaläggningsproblemet och kan lösas med nästan samma giriga algoritm som i boken på sida 116–121/158–163. Den giriga tanken är att den kvarvarande lektion som har den tidigaste sluttiden (om det finns flera med samma sluttid tas vilken som helst) bokas in i det rum som har den hittills största sluttiden som är närmast före den aktuella lektionens starttid, det vill säga det rum som ger det minsta glappet när bokningen görs. Om alla rum är upptagna så slängs lektionen bort (dvs den kommer inte att bli en del av lösningen).

Så här ser algoritmen ut i pseudokod:

ClassroomScheduling(L, k) =
  sortera L med mergesort efter stigande sluttid
  room$\leftarrow$ **new** array$[1 : k]$
  for $j \leftarrow 1$ **to** k **do**

```
    room[j].latest ← 0
    room[j].bookings ← new Queue()
  stoppa in room[1], ..., room[k] i ett rödsvart sökträd RT sorterade efter latest
  for i ← 1  to |L| do
    inv dom första i − 1 lektionerna i L är bokade optimalt enligt room[j].bookings
    hitta det rum r ∈ RT som har största sluttiden (latest) mindre än eller lika med s_i
    if (det finns en sånt rum r) then // låt lektion i bokas in i rum r
      r.latest ← f_i // här måste r flyttas i sökträdet
      r.bookings.Put((s_i, f_i))
    else // släng bort lektionen för det finns inget ledigt rum
  return room
```

Tidskomplexitet: Sorteringen tar tid $O(n \log n)$. Övriga initieringen (fem rader pseudokod) tar tid $O(k \log k)$. Den återstående for-slingan går n varv, och i varje varv görs en sökning i sökträdet och möjligen en flyttning av ett element i sökträdet. Komplexiteten för operationerna på ett rödsvart sökträd är $O(\log k)$, varför komplexiteten blir $O(n \log n) + O(k \log k) + O(n \log k) = O(n \log n)$ eftersom $k < n$.

Undre gräns: För att få en undre gräns för problemet utnyttjar vi undre gränsen på $\Theta(n \log n)$ för att sortera n olika positiva tal med jämförelsebaserad sortering. Detta fungerar eftersom tiderna i indata är godtyckligt stora heltal utan någon speciell fördelning. Låt s_i vara tal nummer i som ska sorteras och låt $f_i = s_i + 1$. Låt $k = 1$, dvs schemalägg i bara ett rum. Om vi löser detta lektionsschemaläggningsproblem så kommer resultatet att vara lektionerna sorterade efter starttid, dvs s_i i stigande ordning, vilket är en lösning till sorteringsproblemet. Eftersom det inte går att sortera snabbare än $\Theta(n \log n)$ så kan inte lektionsschemaläggningsproblemet heller lösas snabbare än så.

Korrekthet: Observera först att enda gången algoritmen INTE bokar in en lektion är då det redan är k inbokade lektioner som överlappar med den förkastade lektionen. Det betyder att det inte går att utöka den lösning som algoritmen ger med någon av dom förkastade lektionerna.

Lemma: Anta att algoritmen står i begrepp att boka in lektionen (s, f) i rum nummer a. Anta att det finns en optimal total lösning som innehåller dom lektioner som algoritmen redan har bokat in, bokade i samma rum som algoritmen bokat. Då finns det en optimal total lösning som både innehåller dom bokningar som algoritmen redan har gjort och dessutom lektionen (s, f) inbokad i rum a.

Bevis av lemmat: Anta att det inte finns en sån optimal total lösning. Då måste det antingen vara så att (s, f) är inbokad i rum $b \neq a$ i den optimala lösningen eller att (s, f) inte är inbokad alls i den optimala lösningen. I det första fallet, eftersom algoritmen placerat in (s, f) i det rum som är ledigt kortast tid före s så måste det vara så att b är ledig längre tid före s än rum a. Det betyder att (s, f) och alla bokningar som görs efter f i rum a lika gärna kan byta plats med dom bokningar som gjorts under samma tid i rum b, och därmed måste det finnas en optimal lösning med (s, f) i rum a.

I det andra fallet, låt (s', f') vara den lektion som i den optimala lösningen är bokad i rum a istället för (s, f). Eftersom (s', f') inte lagts till tidigare i algoritmen så måste $f' \geq f$ så det går bra att ersätta (s', f') med (s, f) och vi har fortfarande en optimal lösning.

Vi ska nu använda ovanstående lemma som induktionssteg i en induktion som startar med tomma bokningsmängden, lägger till bokningar i samma ordning som algoritmen och slutar med den genererade lösningen (dvs invarianten i pseudokoden). Basfallet när det inte finns några bokningar satisfierar trivialt förutsättningarna i induktionslemmat. Använd sedan lemmat i induktionssteget gång på gång tills vi ser att alla lektioner som algoritmen bokar in samtidigt kan ingå i en optimal lösning. Vår första observation visar då att denna lösning måste vara optimal, eftersom lösningen inte kan utvidgas med något av dom önskemål som förkastats av algoritmen.