Eksamen H2020 - IDATT2101

kandidat: 10012

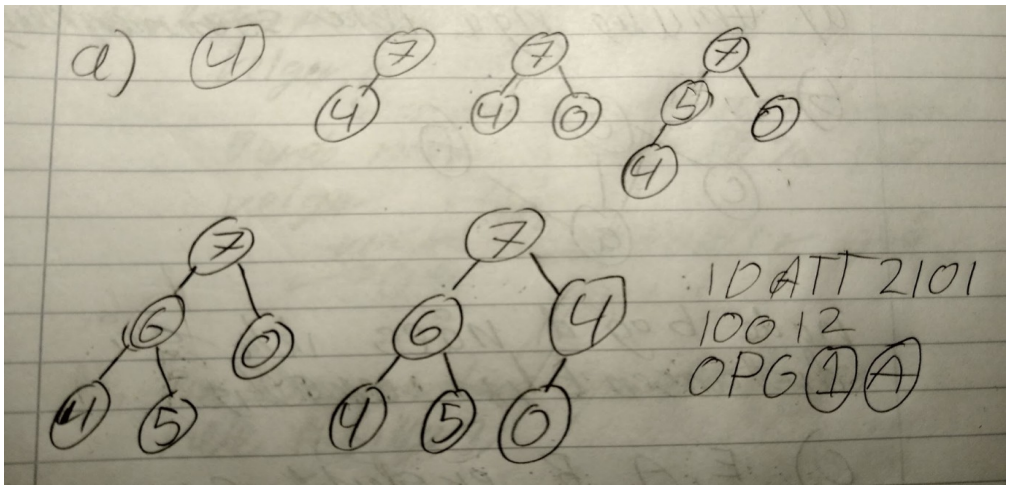
24/11/2020

Oppgave 1

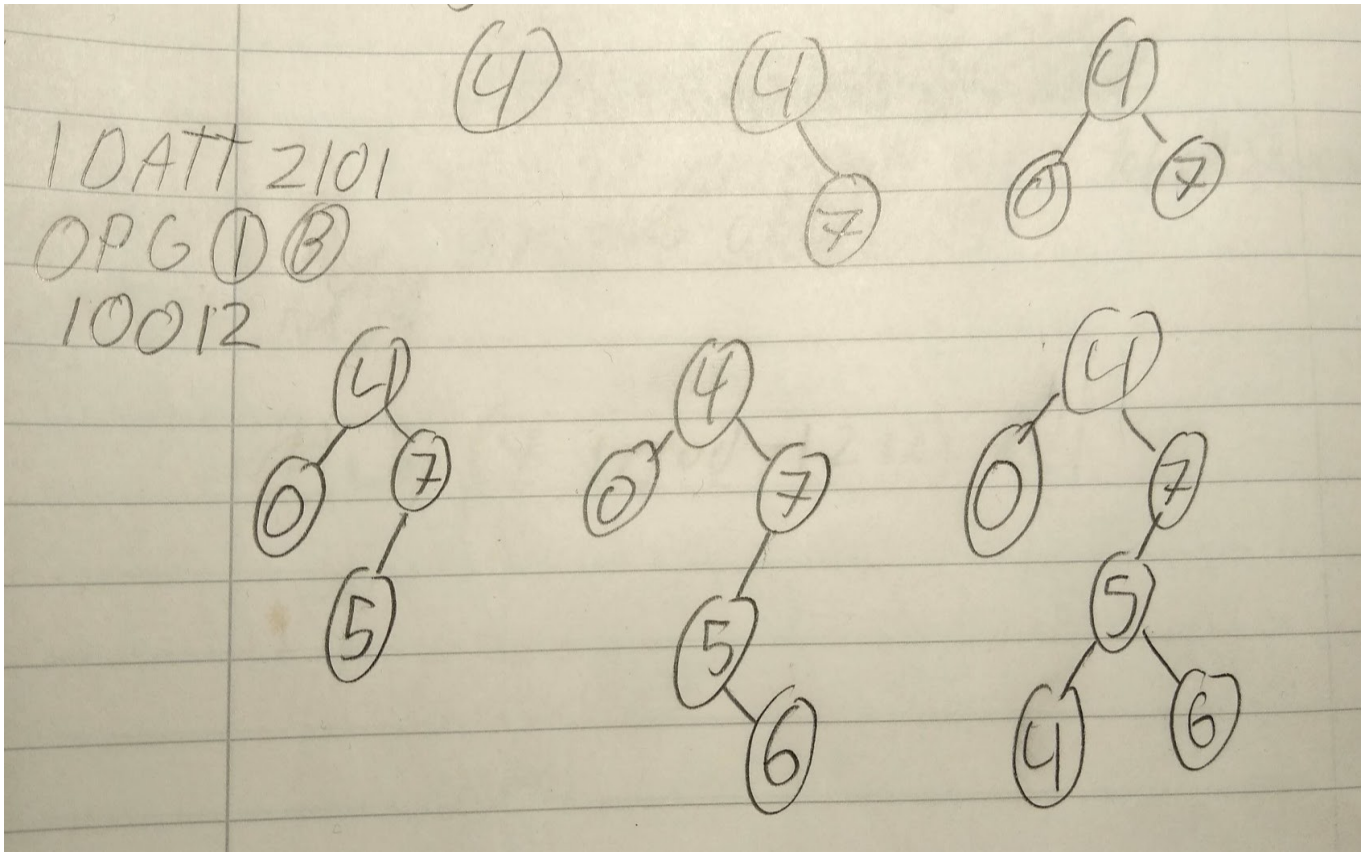
10012 * 47 = 470564

A)

Hver heap er tegnet etter at nødvendige bytter er gjort etter insettingen.



B)
Går her ut ifra at det binære søketreet tillatter like tall, plasserer de i høyre subtre. Hvis de ikke hadde vært tillatt så hadde søketreet vært ferdig i det nest siste trinnet i bildet, da det siste firetallet ikke hadde blitt lagt inn



Oppgave 2
A)
Θ(m^2)

B)
Ω(1), $O(\frac{p}{n})$

C)
T(n) = 1*T(n/2) + n
a = 1, b = 2, k = 1
b^k > a => O(n^k) = O(n)

Ω(1), O(n)

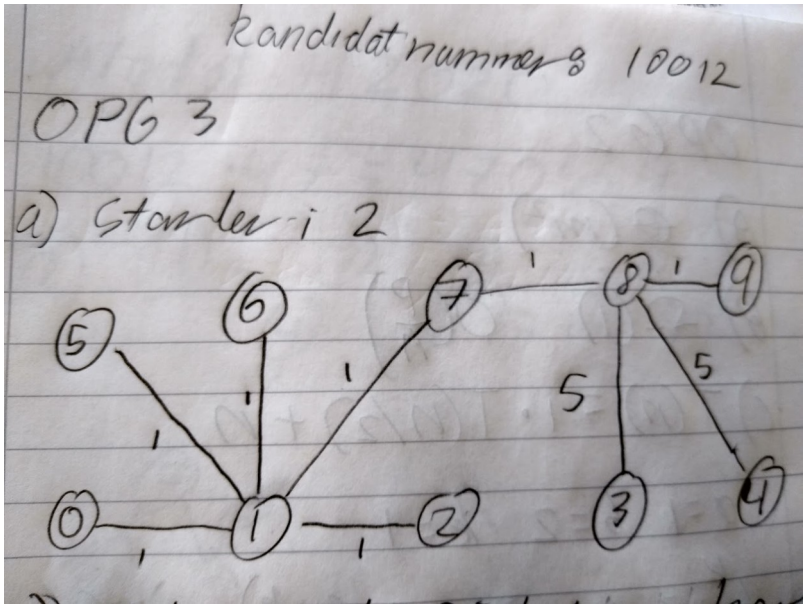
Blir Ω(1) her fordi man skal regne med at alle parametre er større enn eller lik 0. If-sjekken sjekker om m er mindre enn eller lik 0, så selv med kriteriet n >= 0, så kan n være 0, og if-sjekken kan slå inn, noe som gir oss Ω(1)

D)
T(m) = 2*T(m/2) + m^2
a = 2, b = 2, k = 2
b^k > a => O(m^k) = O(m^2)

Ω(1), O(m^2)

Blir Ω(1) her fordi man skal regne med at alle parametre er større enn eller lik 0. If-sjekken sjekker om m er mindre enn eller lik 0, så selv med kriteriet m >= 0, så kan m være 0, og if-sjekken kan slå inn, noe som gir oss Ω(1)

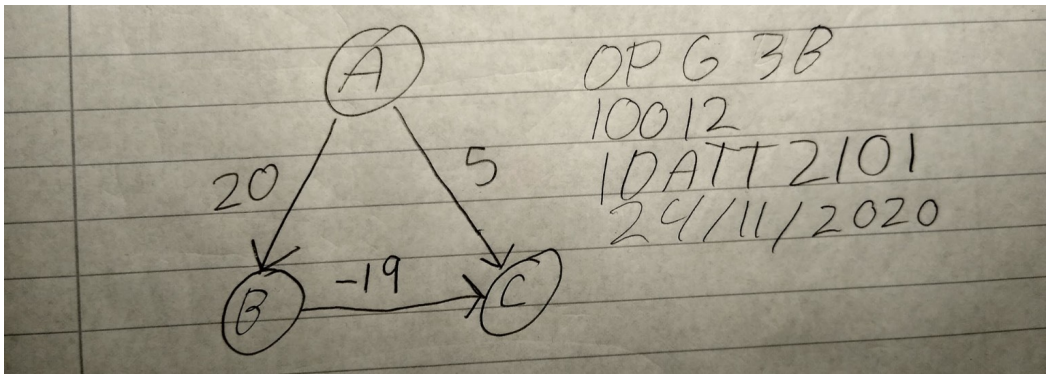
Oppgave 3
A)
Starter i 2



B)
Dijkstra er en grådig algoritme, og tar dermed alltid det valget som ser best ut i øyeblikket, uten å tenke på hvordan det blir senere.

Måten grådigheten er implementert i Dijkstras algoritme er at når en node A har blitt plukket ut fra prioritetskøen så blir A aldri undersøkt igjen, da dijkstra går ut ifra at den korteste veien til A har blitt funnet. Hvis det derimot finnes kanter med negativ vekt inn til A fra en node B som plukkes ut av prioritetskøen senere, så kan det hende at på tross av lengre vei til B enn til A så vil den korteste veien til A gå via B, denne veien vil Dijkstra ikke finne og man får dermed feil svar. I eksempelet nedenfor så kan man se hvordan dette fungerer i praksis

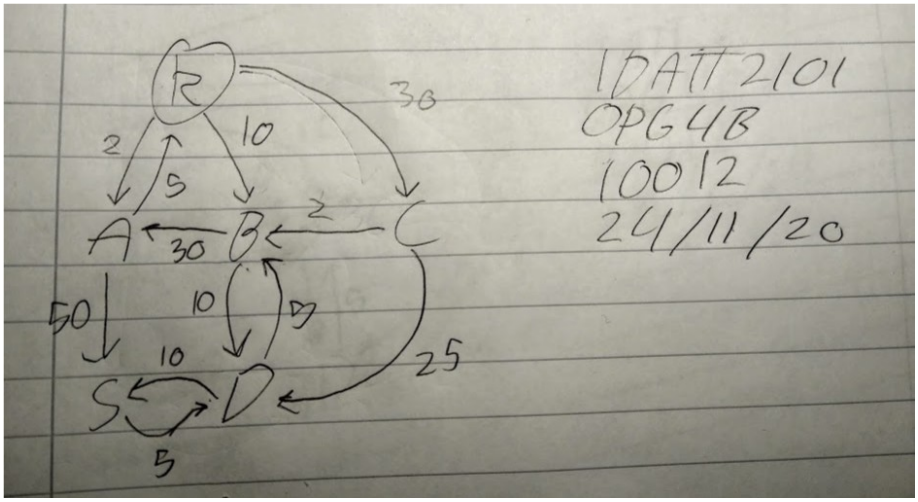
Dijkstras algoritme vil gi feil svar på korteste vei fra A til C til grafen på bildet. Fordi algoritmen er grådig så vil alltid velge å undersøke den korteste veien. Dette fører til at den finner veien A->C først og den går da ut ifra at dette er den korteste veien til C, slik at veien A->B->C ikke blir funnet.



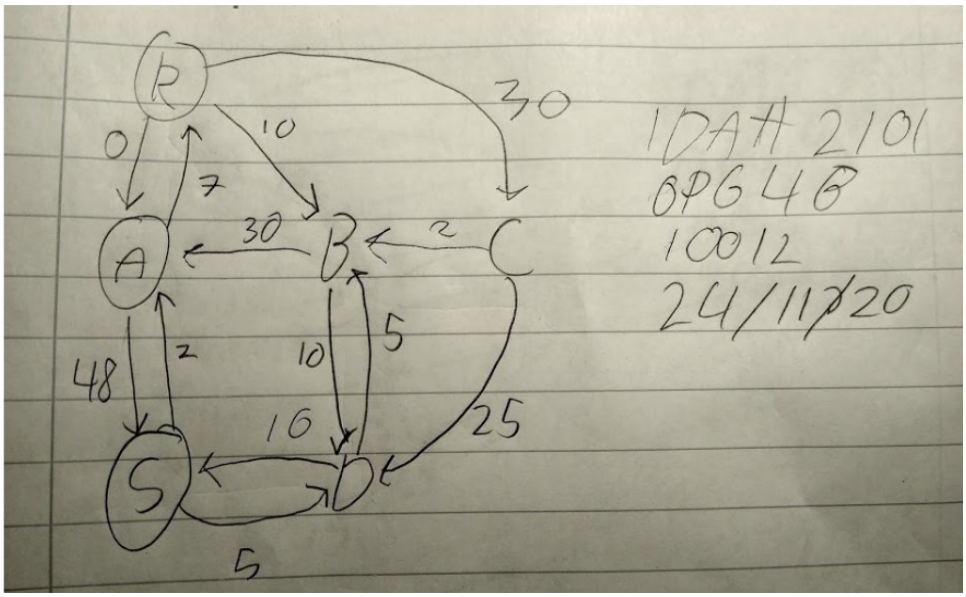
Dette er ikke et problem med kun positive vekter, fordi da vet man at om det er lengre vei til B enn det er til A, så kan ikke den korteste veien til A gå via B.

Oppgave 4

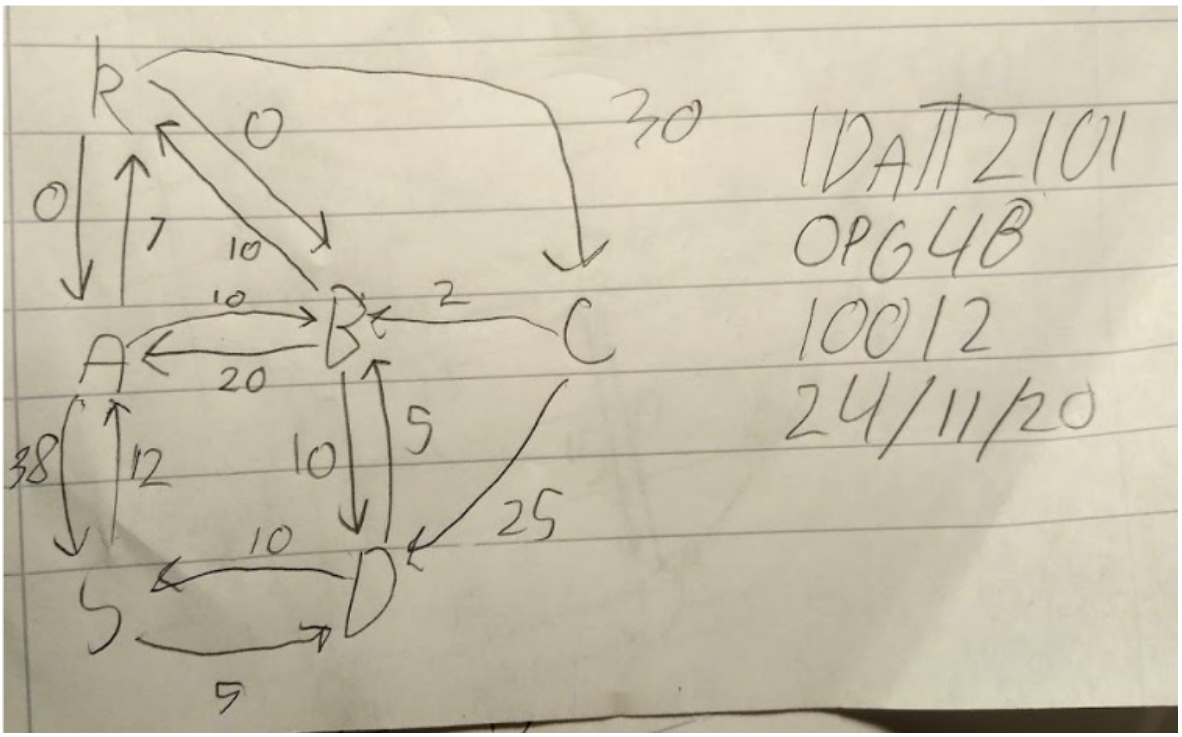
A)
Kandidatnummeret er 10012, så vektene blir 2
den totale vekten til det minimale spenntreet blir 24
B)
Grafen uten endringer:



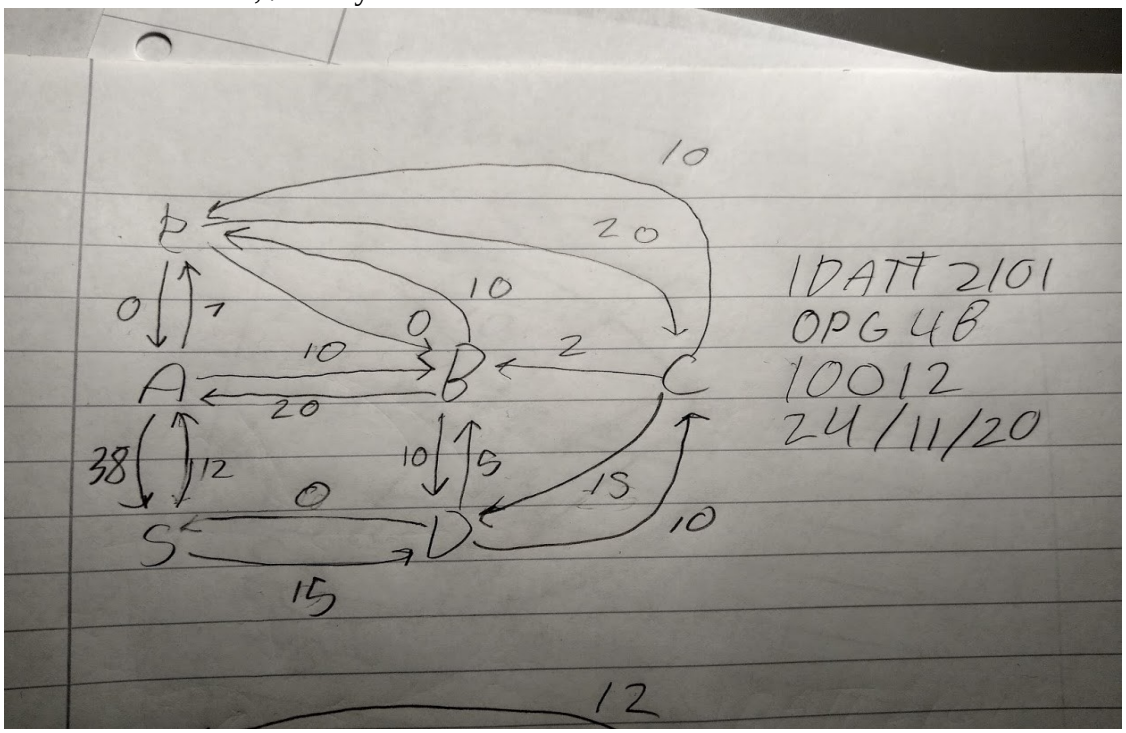
Bruker KAS først, øker flyten med 2, tegner inn negativ flyt som piler i motsatt retning:



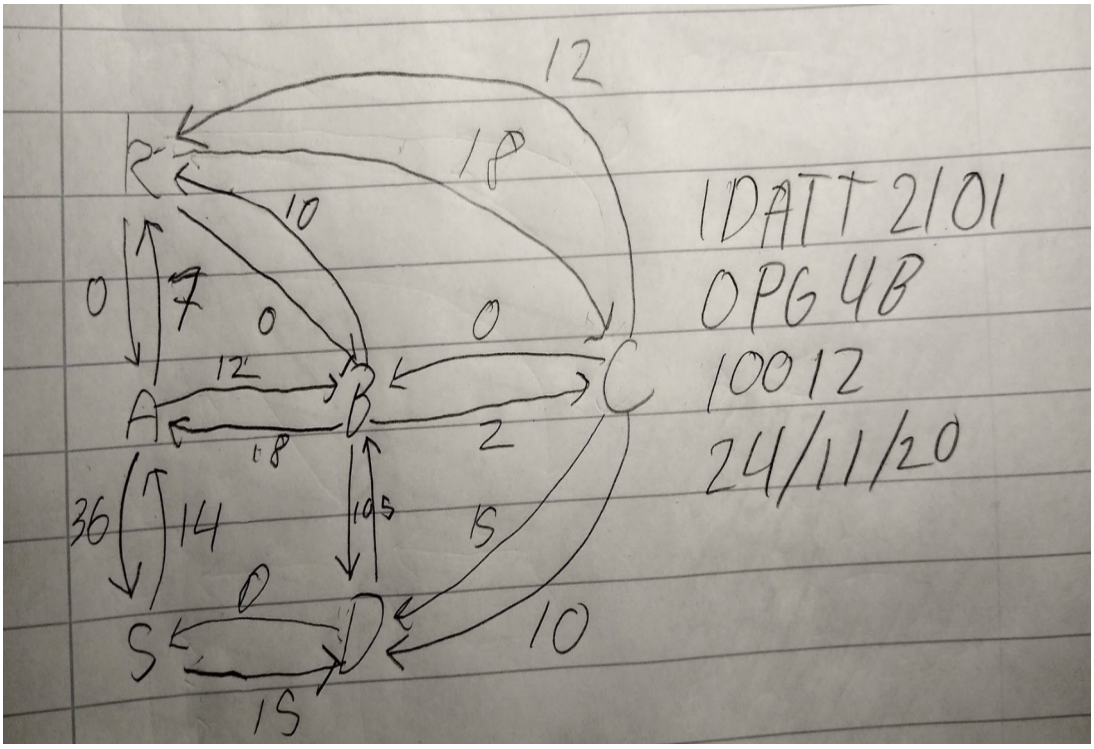
Bruker så KBAS, øker flyten med 10



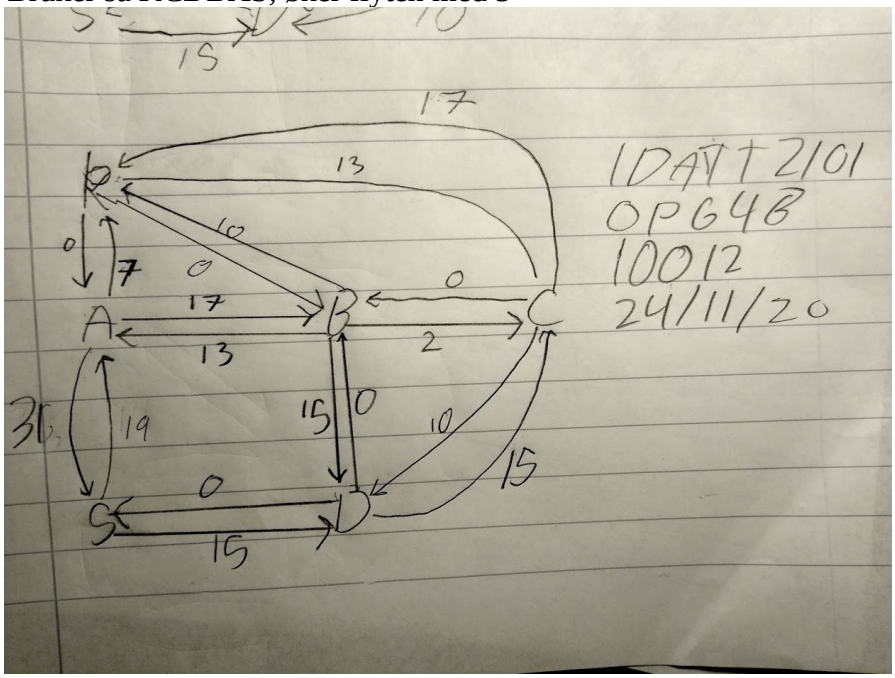
Bruker så KCDS, øker flyten med 10



Bruker så KCBAS, øker flyten med 2



Bruker så KCDBAS, øker flyten med 5



Nå er det ingen flytøkende veier igjen, og den maksimale flyten er dermed 29, man kan finne denne ved å finne differansen i maksimal flyt ut fra S mellom den originale grafen og den siste grafen der jeg har tegnet inn flyt som piler i motstat retning. I den originale så er mulig flyt ut fra S: 5, i den siste grafen så så er mulig flyt ut fra S: 34
Den totale flyten fra K til S blir dermed 34 – 5 = 29.

Oppgave 5
A)
Det at et problem er i kompleksitetsklassen P vil si det kan løses i polynomisk tid. Et eksempel er sortering. For å sortere en rekke tall så er man ikke nødt til å prøve alle mulige kombinasjoner, som ville ha vært O(n!). Istedenfor så kan man bruke en algoritme som merge sort, som har en tidskompleksitet på O(n*logn), noe som er polynomisk tid. Man kan se at dette er polynomisk tid ved: $O(n*\log(n))\in O(n^2)$
Dette er mye kjappere enn å prøve alle mulige permutasjoner. Polynomisk tid er O(n^k) hvor k er et positivt heltall eller 0.

B)
Det at et problem er i NP betyr at et gitt svar kan sjekkes i i polynomisk tid, men det trenger ikke å nødvendigvis kunne løses like raskt. Alle problemer i P er også i NP, $P\in NP$.

Et eksempel på et problem i NP som ikke er i P er travelling salesman-problemet, hvor vi har en gruppe byer og reisekostnader mellom de. Vi ønsker å finne en rute som er innom alle byene med en kostnad under x. For å finne en løsning så må man prøve forskjellige ruter helt til man finner en med lav nok kostnad. For å sjekke en løsning så kan man simpelthen summere kostnadene til alle veiene og sjekke om summen er under x. Dermed så er det å sjekke en mulig løsning O(n)

c)
Når man hasher med restdivisjon så må man bruke modulo av tabellengden, hvis ikke så vil man få ubrukte indekser i slutten av tabellen, så jeg går ut ifra at Kåre har valgt en løsning som vil unngå dette:
h(k) = k % 100
Hvor k er et gitt postnummer.

Når man bruker restdivisjon med en tierpotens 10^x hvor x er positivt heltall, så vil svaret kun avhenge av de x siste sifrene i svaret.
I eksempelet så er restdivisjon derfor en dårlig løsning, da det står at mange postnumre slutter på 0. Det er derfor lite variasjon i de to siste sifrene, og det vil føre til at hash-verdiene vil avhenge av kun ett siffer i mange av postnumrene. Kåre vil derfor få svært mange tall som hasher til tall i 10-gangen, noe som vil føre til unødvendig mange kollisjoner.

En bedre løsning er å velge en tabellstørrelse som er et primtall. Dersom man da brukte hashing med restdivisjon så vil hashverdien variere basert på alle sifrene i postnummeret, ikke kun de to siste.

Hvis man hadde ønsket å beholde tabellstørelsen så kunne man ha brukt multiplikasjon med desimaltall. Man kunne ha brukt funksjonen: $h(k) = \lfloor 100 * (k * A - \lfloor k * A \rfloor) \rfloor$ hvor k er postnummeret man vil hashe og A er et passende desimaltall, f.eks. $\frac{(\sqrt{5}-1)}{2}$