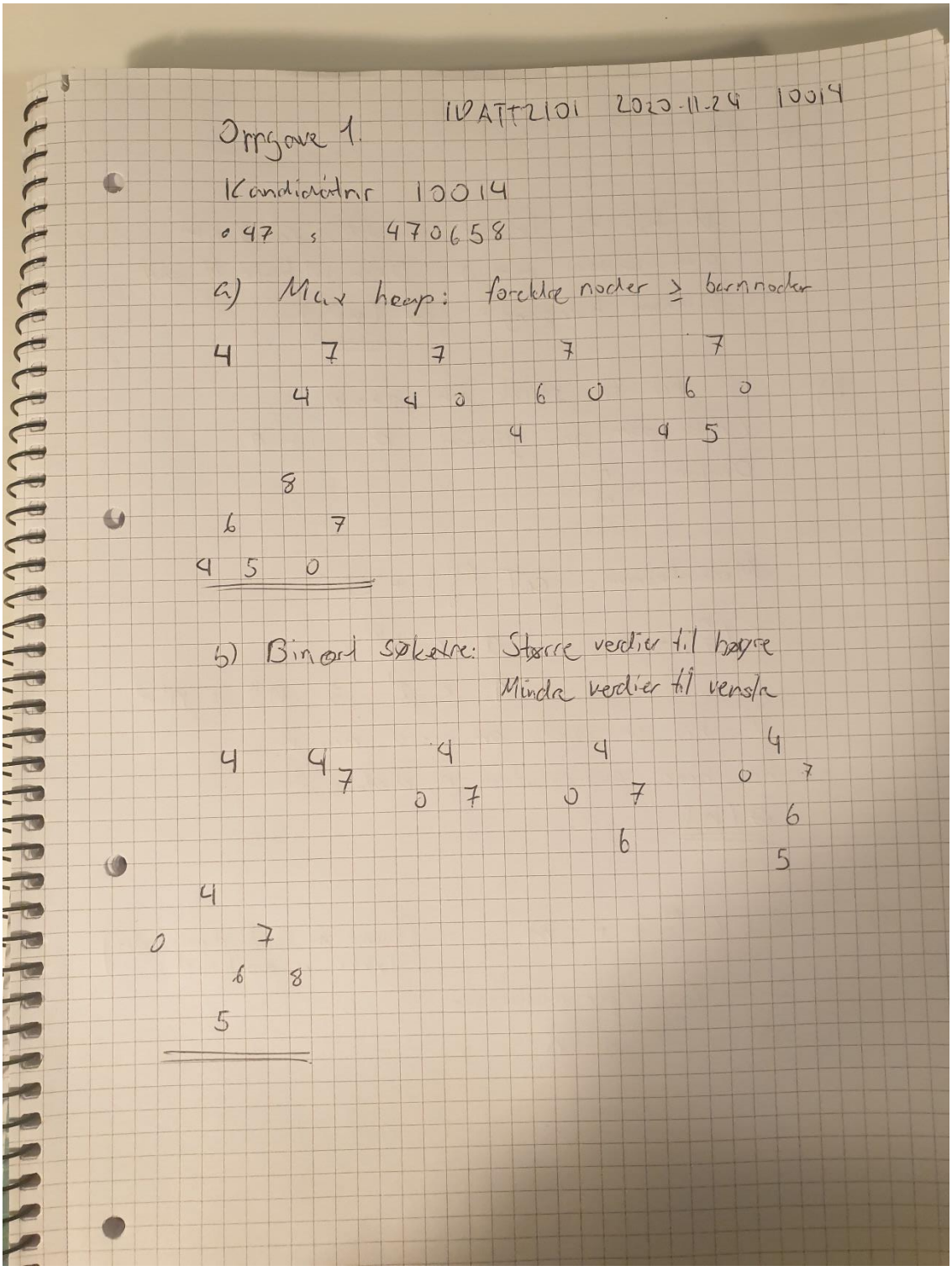
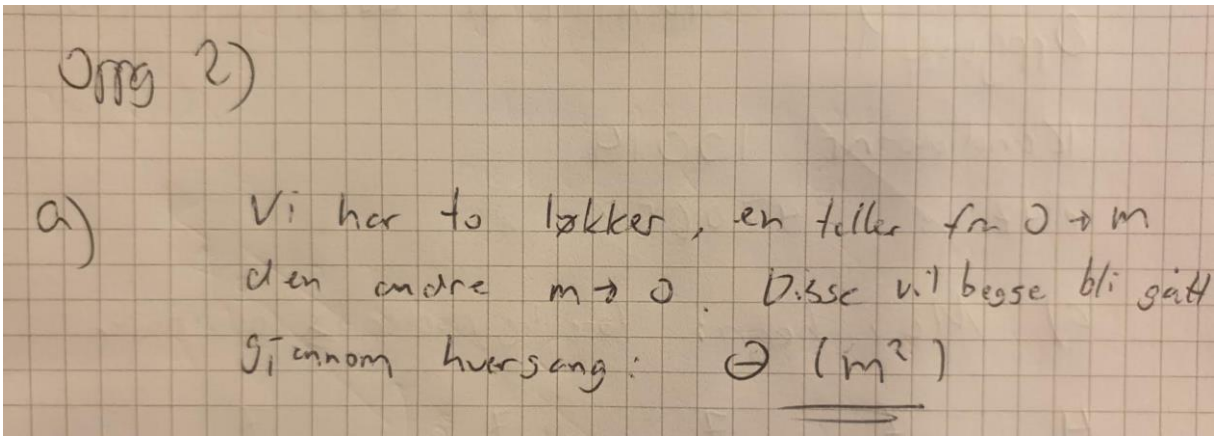
Oppgave 1 a og b

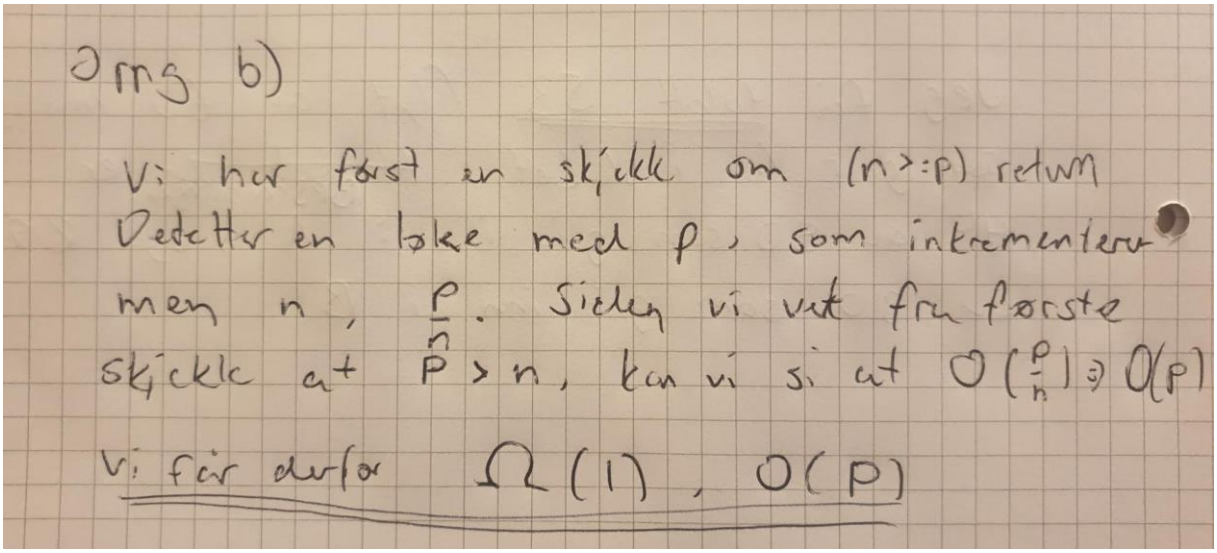


Oppgave 2

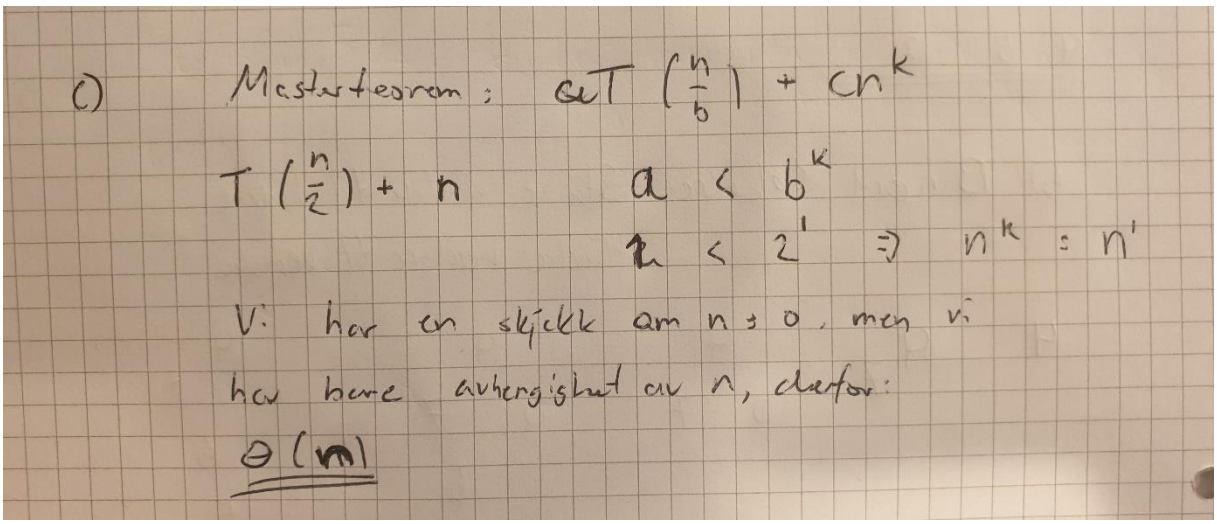
Oppgave a: $\Theta(m^2)$



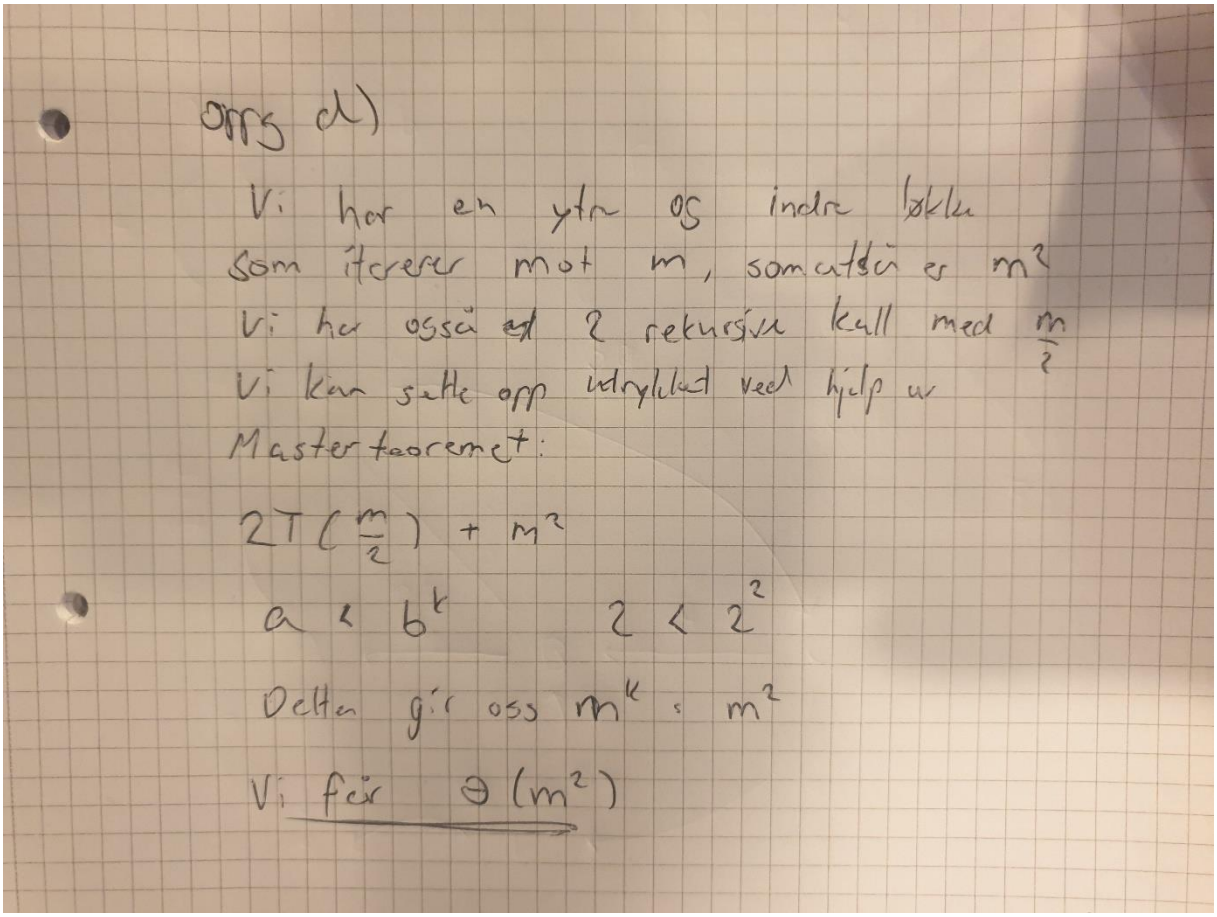
Oppgave b: Ω (1), O(p)



Oppgave c: $\Theta(n)$

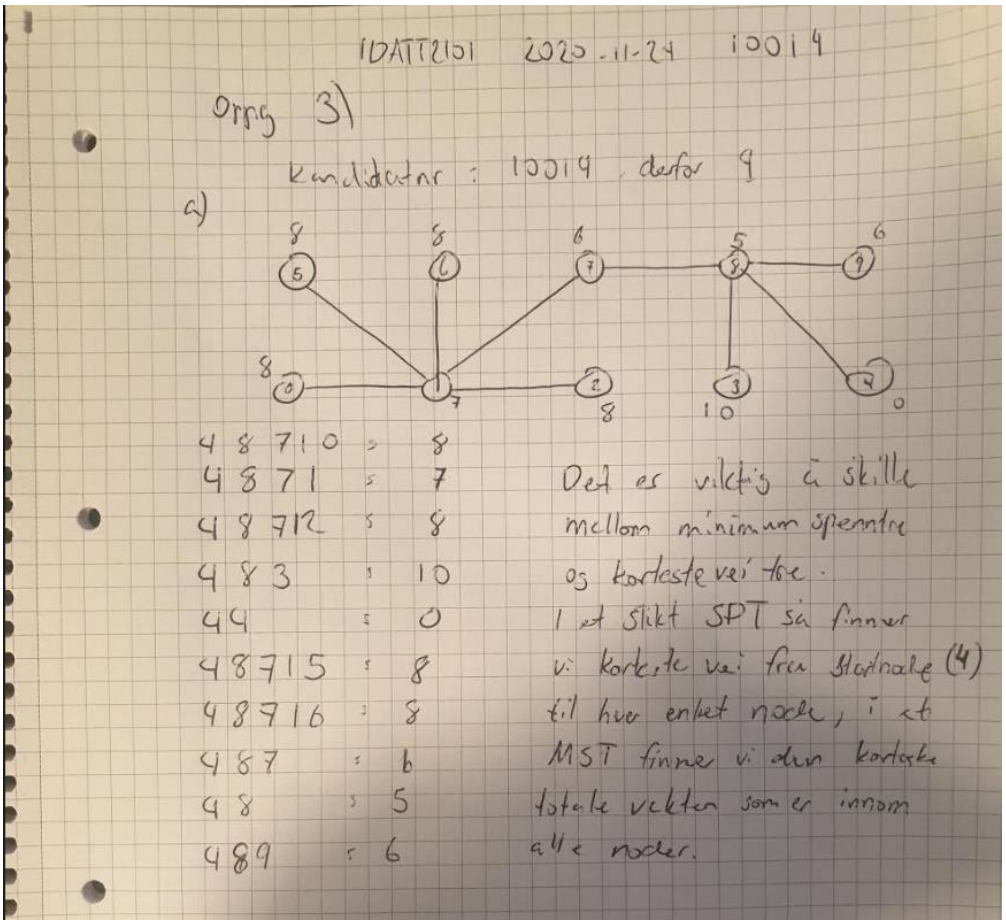


Oppgave d: $\Theta(m^2)$



Oppgave 3

Oppgave a



Det kan finnes flere SPT, her har vi et annet med 4-9 i stedet for 4-8-9, begge er 6 lang

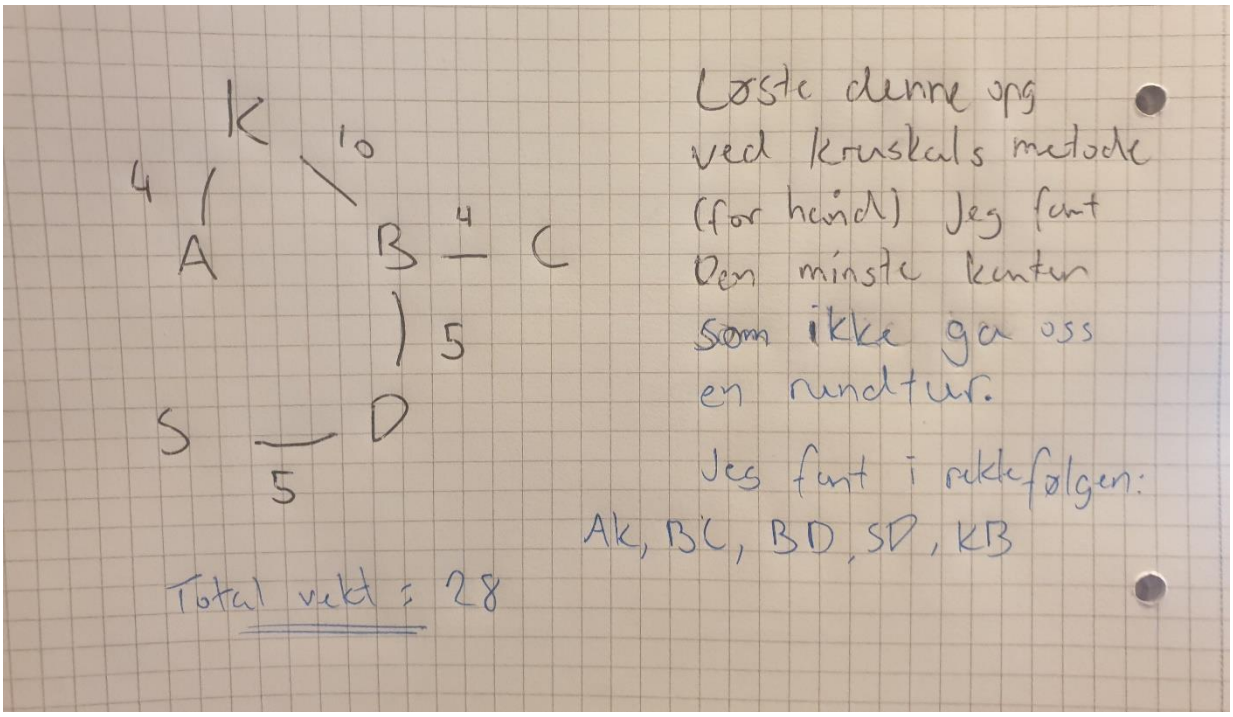
Oppgave b

Dijkstra er en grådig algoritme som velger den veien som den tror er best her og nå, ved en Prioritetskø som prioriterer på vekt, slik at vi alltid kan velge den veien med lavest total distanse fra startnoden. Dette fungerer i tilfeller med kun positive kantvekter (den kan gi riktig svar i tilfeller med negative kanter, men det vet vi i så fall ikke). Dijkstra kan plukke ut noder fra prioritetskøen når noden er sjekket, fordi den da nødvendigvis vil ha funnet den korteste veien (siden alle andre veier den som den kan finne i ettertid vil være kortere). Om vi derimot legger inn negative kanter, kan det være slik at det finnes en alternativ vei som har en negativ vekt, slik at den totale vekten blir lavere enn den som originalt var funnet. Siden Dijkstras opererer grådig, og har plukket ut av prioritetskøen, vil den ikke ha noen mulighet til å oppdage denne kortere veien.

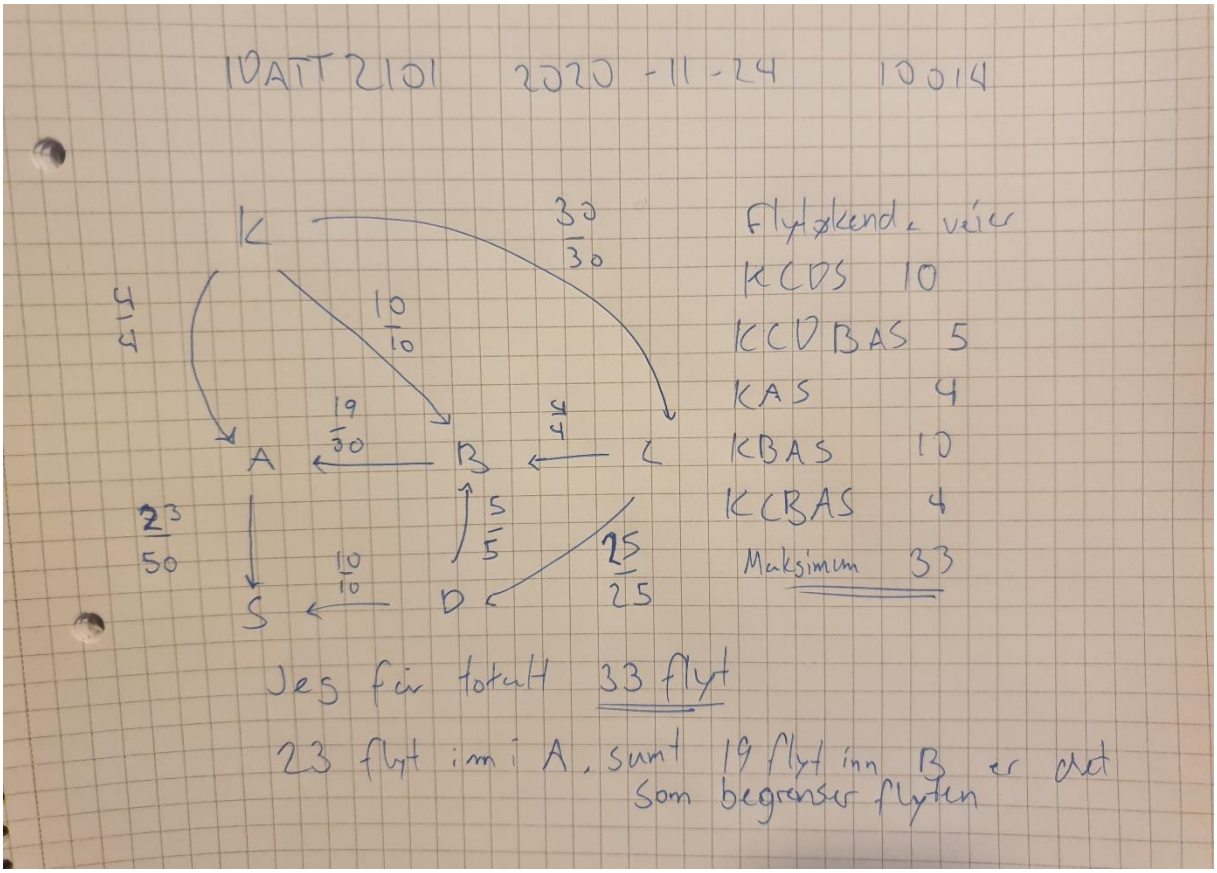
Alternativt kan man bruke Bellman Ford, som håndterer negative kanter, og for øvrig gir tilbakemelding dersom vi har en negativ rundtur (som ødelegger da man bare kan kjøre den samme rundturen flere ganger og minke totalvekt). Den har derimot lenger kjøretid sammenlignet med Dijkstra.

Oppgave 4

Oppgave a: Total vekt 28



Oppgave b, Maksimal flyt = 33



Oppgave 5

Oppgave a

Et problem som inngår i kompleksitetsklassen P, har et svar vi kan løse, og denne løsningen kan vi finne i polynomisk tid. Siden P inngår i NP må vi nødvendigvis også kunne vise at et svar er riktig i polynomisk tid. P er ethvert problem som kan løses i polynomisk tid, er slikt problem vi har jobbet med er korteste vei problemet.

Oppgave b

Et problem som er i kompleksitetsklassen NP har et svar som vi kan vise er riktig i polynomisk tid, men i motsetning til P trenger man ikke nødvendigvis å kunne løse dette problemet i polynomisk tid. Alle P inngår i NP.

Vi kan også utvide definisjonen, og inkludere NP-komplette og NP-harde problem. NP-komplette problemer er spesielt vanskelige problemer i NP, men disse er som sagt løsbare med en løsning som kan sjekkes i polynomisk tid. NP-harde problem trenger derimot ikke å kunne sjekkes i polynomisk tid heller (noen problemer er også uløselige, slik som Halting problemet som gir oss en selvmotsigelse).

Vi har det NP-komplette problemet Traveling Salesman, hvor man skal finne en vei som koster mindre enn x. Dette er vanskelig og kan ikke løses i polynomisk tid, men den er derimot lett å verifisere dersom vi har fått et svar. Om vi derimot sier vi skal ha den korteste ruten, er det ikke lenger like enkelt, og vi kan bevise at det er riktig i polynomisk tid. Dette blir NP-hard varianten av problemet, og jeg liker å tenke på det som at du må løse problemet i seg selv, for å bevise at løsningsforslaget du har fått er riktig.

Oppgave c

Lærerboka Algoritmer og Datastrukturer (Hafting H, Ljosland M) diskuterer restdivisjon på side 157, og sier at det vil være problematisk å bruke restdivisjon på tabellstørrelse m = 10^x. Siden m i dette tilfelle er 100, altså en tierpotens, vil det ikke være gunstig med restdivisjon.

I dette spesifikke tilfelle har vi postnummer, og her vil hashfunksjonen vår avhenge kun av de to siste sifrene av nøkkelen. Om vi altså har Trondheim 7030, vil det kun være 30 som har innflytelse på nøkkelen vi får, med tierpotens vil det bare avhenge av de siste sifrene. Vi ønsker at hashfunksjonen vår skal være avhengig av alle deler av nøkkelen vår. Dette kan vi gjøre ved å benytte oss av et primtall i stedet. Hadde altså tabellstørrelsen m vært et primtall, kunne restdivisjon vært gunstig.