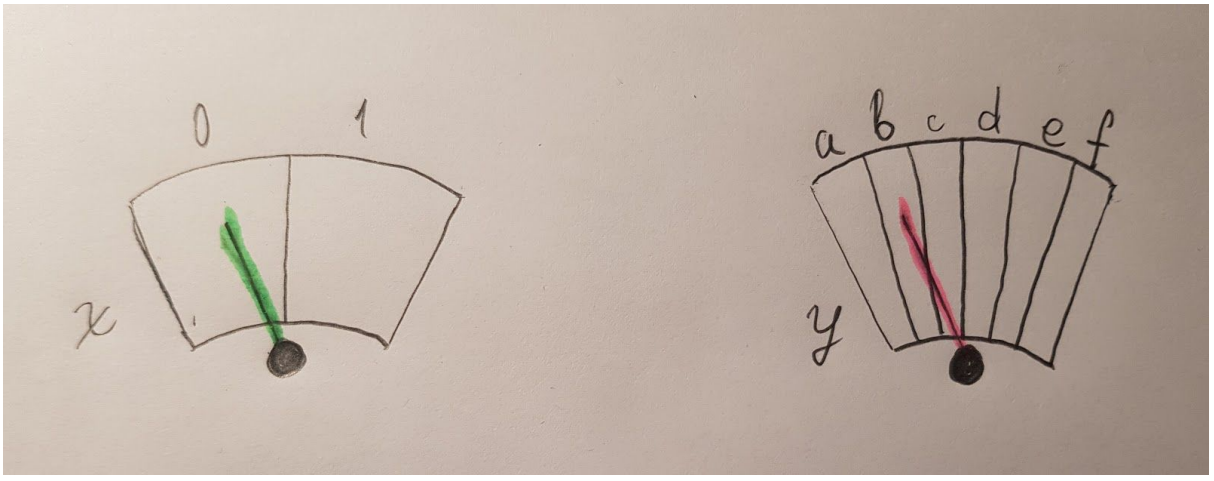
Oppgave 1

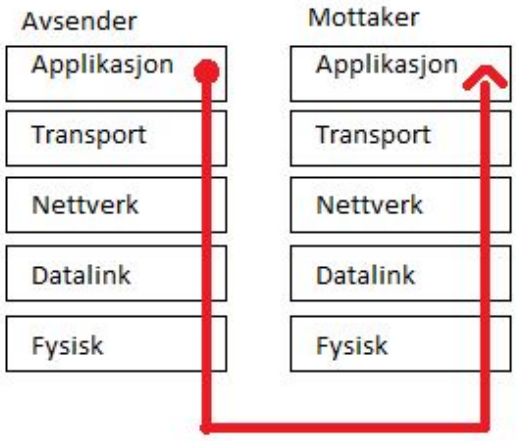
a)

Binær logikk brukes i datamaskiner på grunn av dens enkelhet og pålitelighet når det kommer til konstruksjon av elektroniske kretser. En krets med kun to tilstander (på og av, eller 1 og 0) fører til mindre sjanse for feil ved avlesning enn en krets med f.eks. 6 tilstander (se figur). Kretser som bruker binær logikk kan også kode det meste uten store vansker; unntaket er kanskje flyttall, som krever en egen standard (IEEE 754) for å fungere.



Figur *x* skal vise 0; på grunn av en liten feil i systemet er spenningen ikke helt i bunn, men leseren leser fortsatt av 0.
Figur *y* skal vise a; på grunn av samme spenningsfeil viser leseren nå b istedenfor.
Med dette kan vi trekke konklusjonen at muligheten for feil er mindre ved færre tilstander. En enklere krets vil også være billigere å produsere.

b)



Applikasjonslaget (5):

- Hovedoppgave: Støtte nettverksapplikasjoner som f.eks. sender/mottar epost, eller laster ned filer.
- Eksempel på protokoller: HTTP, SMTP, FTP.

Transportlaget (4):
- Hovedoppgave: Transportere segmenter mellom tjener og klient i applikasjonslaget.
- Eksempel på protokoller: UDP, TCP.

Nettverkslaget (3):
- Hovedoppgave: Route datagram fra avsender til mottaker.
- Eksempel på protokoller: IP, ICMP.

Datalinklaget (2):
- Hovedoppgave: Overføre ramme mellom nabonoder (switcher og routere).
- Eksempel på standarder: Ethernet II, FDDI.

Det fysiske laget (1):
- Kabler, antenner, nettverkskort, etc.

Man starter med en melding i applikasjonslaget hos avsenderen.
I transportlaget får meldingen en transportheader, avhengig av protokoll. Denne inneholder bl.a. avsenderens og mottakerens portnummer. I transportlaget kalles det da segment.
Nettverkslaget legger så på en IP-header, som bl.a. inneholder avsenderens og mottakerens IP-adresse. Nettverkslagets overføringsenhet kalles datagram.
Datalinklaget legger så på enda en header, med avsenderens og mottakerens MAC-adresser. Dette laget legger også til en trailer med feilsjekkingsinformasjon. Overføringsenheten her kalles ramme.
Informasjonen blir så fraktet over det fysiske laget og “pakkes ut” i motsatt rekkefølge hos mottakeren, hvor hvert lag fjerner relevant header/trailer.

c)
Et operativsystem (OS) er “mellommann” mellom maskinvare og bruker/applikasjon. Sett fra maskinvaresiden er OS det som administrerer tids- og plassbruk av CPU og minne. Sett fra brukersiden er OS noe som abstraherer bort de tyngre oppgavene, og lar bruker/utvikler ha noe mer oversiktlig å jobbe i/med.

User mode er en operasjonsmodus i OS som kun har tilgang på et begrenset sett instruksjoner (maskinkode) som kan kjøres på CPU, og kun det minnet OS har tildelt. User mode har dessuten ikke tilgang på maskinvare direkte, men kan kun kommunisere med det via et API (Application Programming Interface). Slik jeg forstår det, er user mode ment å være en beskyttende modus, som ikke kan gjøre for stor skade på maskinen om noe skulle gå galt.

Kernel mode har derimot tilgang til hele minnet og alle CPU-instruksjoner, og dermed også maskinvare.

Oppgave 2

a)

Deler først tallet opp i toerpotenser:

$495_{10} = 256 + 128 + 64 + 32 + 8 + 4 + 2 + 1$
$= 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$

Skriver det så om til binær, 16 bits presisjon:

$= 0000\ 0001\ 1110\ 1111_2$

Bruker samme fremgangsmåte på neste deloppgave:

$55_{10} = 32 + 16 + 4 + 2 + 1$
$= 2^5 + 2^4 + 2^2 + 2^1 + 2^0$
$= 0000\ 0000\ 0011\ 0111_2$

b)

$$\begin{array}{r} \ 1\ \ \ 1 \\ 0111\ 0101_2 \\ \underline{+\ 0010\ 0110_2} \\ = 1001\ 1011_2 \end{array}$$

$$\begin{array}{r} \ 11 \\ 1011\ 0001_2 \\ \underline{+\ 1110\ 0100_2} \\ = 1001\ 0101_2 \end{array}$$

Tar ikke med overflow, pga 8 bits presisjon.

c)

Bruker 16 bits presisjon:

$301_{10} = 256 + 32 + 8 + 4 + 1$
$= 2^8 + 2^5 + 2^3 + 2^2 + 2^0$
$= 0000\ 0001\ 0010\ 1101_2$

$100_{10} = 64 + 32 + 4$
$= 2^6 + 2^5 + 2^2$
$= 0000\ 0000\ 0110\ 0100_2$

Finner toerkomplement av $0000\ 0000\ 0110\ 0100_2$:
Flip/enerkomplement:

$1111\ 1111\ 1001\ 1011$

+1:

$$\begin{array}{r} \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 11 \\ 1111\ 1111\ 1001\ 1011_2 \\ \underline{+\ 0000\ 0000\ 0000\ 0001_2} \\ = 1111\ 1111\ 1001\ 1100_2 \end{array}$$

Regner ut:

```
  1111 111
  0000 0001 0010 1101₂
+ 1111 1111 1001 1100₂
= 0000 0000 1011 1101₂
```

Tar ikke med overflow pga 16 bits presisjon.

d)

Unicode-standarden bygger på en struktur hvor man har 17 plan (0-16), med 65536 kodepunkter på hvert plan. Hvert kodepunkt har en 7-21 bit lang binærkode, oftest skrevet i hex-format på denne måten: U+0041 (kodepunktet for "A"). Plan 0 begynner med samme struktur som ASCII, og er ment å inneholde tegn som brukes i "vanlige" språk. Plan 1 er delvis supplementær til plan 0, men inneholder også spesielle tegn som hieroglyfer. Emojis er også å finne her. Plan 2 er mindre brukte kinesiske tegn. Plan 3 inneholder sjeldne og historiske kinesiske, japanske, koreanske, og vietnamesiske tegn. Før mars 2020 hadde ikke plan 3 noen tildelte tegn, slik plan 4-13 er nå. Plan 14 er spesialtegn for bl.a. tagging av språk, mens plan 15-16 er til privat bruk (firmalogoer, o.l.).

Unicode kan enkodes ved hjelp av transformasjonsformater som f.eks. UTF-16 og UTF-8. UTF-8 er i stor grad standarden på Internett i dag. Det er verdt å nevne at Microsoft har vært spesielt lite nøye på bruk av riktig terminologi når det kommer til enkoding av Unicode i UTF-16. I tidligere versjoner av Windows, når man lagret en fil i Notepad, ville man bl.a. få valget mellom enkodingene Unicode og Unicode Big Endian. Med dette mentes det UTF-16 LE og UTF-16 BE.

e)

Gjør først hex-tallet 0x2655 (U+2655) til binært:

```
  2    6    5    5
0010 0110 0101 0101
```

Ser at tallet må ha 16 bits, bruker derfor 3 bytes til enkoding:

```
1110 0010 1001 1001 1001 0101
```

Konverterer så til hex:

```
1110 0010 1001 1001 1001 0101
  E    2    9    9    9    5
```

Kodepunktet U+2655 i Unicode kodes som 0xE29995 i UTF-8

f)

Konverterer først begge hex-tall til binært, utfører så operasjonen, og konverterer tilbake til slutt. Følger samme fremgangsmåte på alle deloppgaver:

```
0x42 & 0xF2

  4    2
0100 0010

  F    2
1111 0010

  0100 0010
& 1111 0010
= 0100 0010

0100 0010
  4    2

= 0x42
```

```
0x39 & 0xAC

  3    9
0011 1001

  A    C
1010 1100

  0011 1001
& 1010 1100
= 0010 1000

0010 1000
  2    8

= 0x28
```

```
0x55 | 0x72

  5    5
0101 0101

  7    2
0111 0010

  0101 0101
| 0111 0010
= 0111 0111

0111 0111
  7    7

= 0x77
```

```
0xFF | 0x34

= 0xFF
```

Alle OR-operasjoner hvor et av input'ene består av kun enere vil returnere kun enere; denne oppgaven trenger ikke utregning. Jeg viser den likevel, for sikkerhets skyld:

```
  F    F
1111 1111

  3    4
0011 0100

  1111 1111
| 0011 0100
= 1111 1111

1111 1111
  F    F

= 0xFF
```

```
0x69 ^ 0xBB

  6    9
0110 1001

  B    B
1011 1011

  0110 1001
^ 1011 1011
= 1101 0010

1101 0010
  D    2

= 0xD2
```

g)

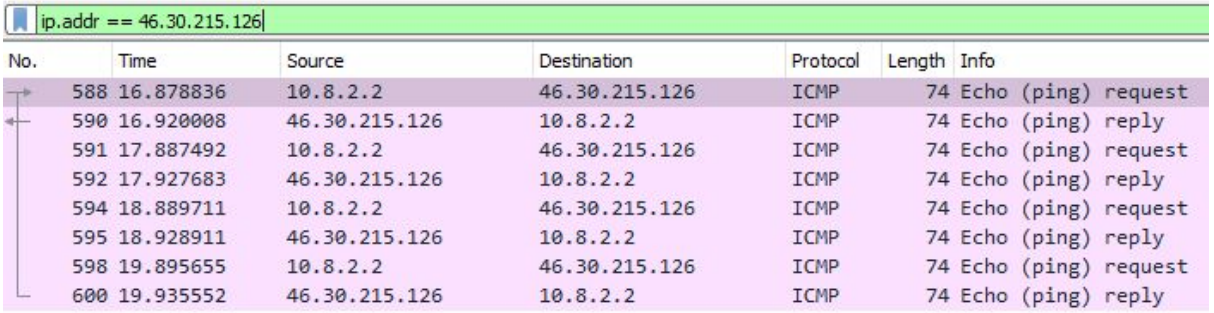
$1814_{21} = 1 * 21^3 + 8 * 21^2 + 1 * 21^1 + 4 * 21^0$
$= 9261 + 3528 + 21 + 4$
$= 12814_{10}$

Oppgave 3

Alle deloppgaver er løst i Windows

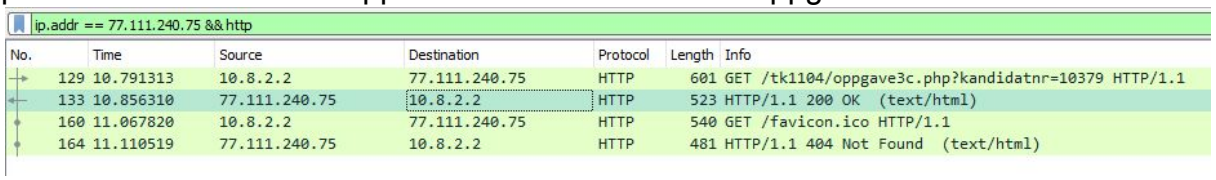
a)

"`ping www.h-ck.me > list`" sjekker hvor lang round-trip-time (sende forespørsel og få svar) er mellom brukerens maskin og IP-adressen www.h-ck.me tilhører, og legger output av denne ping-request'en i en fil som heter "list". Filen vil ligge i mappen man kjører kommandoen fra, som i mitt tilfelle vil være `C:\Users\[brukernavn].`
Ping-kommandoen bruker ICMP-protokollen for å både sende ping requests og motta ping replies. I mitt tilfelle ble det sendt og mottatt fire requests/replies, uten tap:



b)
Kommandoen `netstat -r` i CMD vil vise datamaskinens routing-tabell (både IPv4 og IPv6). `netstat` brukes generelt for å se nettverksstatus (derav navn på kommando). Ved å kun kjøre `netstat`, uten `-r`, vil man f.eks. kunne se nåværende åpne forbindelser, og hvilken protokoll de bruker.

c)
Jeg begynte med å kjøre `ping`-kommandoen i CMD mot www.eastwillsecurity.com, som ga meg en IP-adresse (77.111.240.75) jeg kunne filtrere trafikken i Wireshark gjennom senere. Deretter startet jeg opp Wireshark og lot verktøyet fange opp trafikk, og skrev "http://www.eastwillsecurity.com/tk1104/oppgave3c.php?kandidatnr=10379" inn i adressefeltet i Chrome. Deretter satte jeg filteret "ip.addr == 77.111.240.75 && http" på trafikken for å kun få opp det som var relevant for oppgaven:



Jeg så nærmere på den andre pakken ovenfra (merket med lysegrønn), og merket meg disse headerne:

```
Date: Tue, 15 Dec 2020 12:33:03 GMT\r\n
Server: Apache\r\n
X-Powered-By: PHP/7.4.13\r\n
Expires: 0\r\n
Cache-Control: must-revalidate\r\n
X-Hash: 472047204000010379561763 98\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Content-Length: 77\r\n
Content-Type: text/html;;charset=UTF-8\r\n
X-Varnish: 141332349\r\n
Age: 0\r\n
Via: 1.1 varnish (Varnish/6.5)\r\n
Accept-Ranges: bytes\r\n
Connection: keep-alive\r\n
```

Jeg vil anta `\r\n` på slutten av hver linje betyr `return` og `linefeed`, og refererer til `cr` og `lf` (carriage return, line feed), respektivt.

Jeg velger også å overse den andre HTTP-pakken jeg mottok (fjerde linje i bildet), da denne er et svar på en GET-request for å få et ikon som tilhører nettsiden.

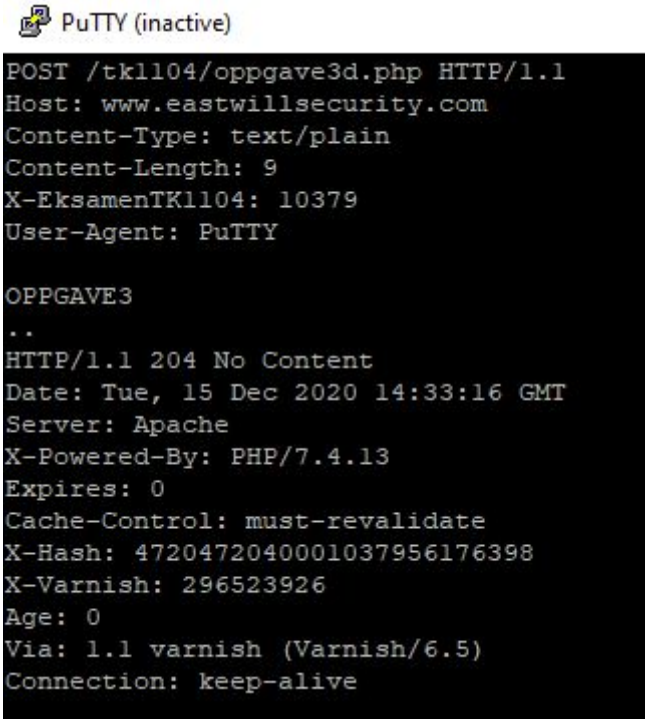
Innholdet på nettsiden var teksten "Html response for oppgave 3c".

d)
Kjørte først PuTTY som administrator, og satt "host name" til å være www.eastwillsecurity.com, "port" til 80, "connection type" til Raw, og "close window on exit" til never. Deretter skrev jeg følgende inn på Notepad:

```
POST /tk1104/oppgave3d.php HTTP/1.1
Host: www.eastwillsecurity.com
Content-Type: text/plain
Content-Length: 9
X-EksamenTK1104: 10379
User-Agent: PuTTY

OPPGAVE3
..
```

Jeg kopierte denne teksten, og åpnet så tilkoblingen i PuTTY. Der limte jeg inn teksten, og fikk dette til svar (les fra linje 10):



Grunnen til at header'ne `Content-Type` og `Content-Length` er satt til å være `text/plain` og `9`, respektivt, er at meldingen jeg sender er ren tekst med ni tegn (åtte tegn + ny linje). Uten disse header'ne oppfatter ikke mottaker at meldingen inneholder en body.

Oppgave 4

a)
ARPANET (Advanced Research Projects Agency Network) var et amerikansk nettverk av datamaskiner i som ble forløperen til dagens Internett. ARPANET ble etablert i 1968 av organisasjonen ARPA (senere DARPA, Defense Advanced Research Projects Agency), etter Bob Taylors initiativ. Nettverket skulle knytte sammen amerikanske universiteter, og diverse føderale organer (f.eks. Pentagon), og i 1973 ble også det norske NORSAR (Norwegian Seismic Array) tilknyttet prosjektet.

ARPANET var det første nettverket som benyttet seg av pakkesvitsjing istedenfor linjesvitsjing, som gjorde det mulig å benytte seg av 100% av den tilgjengelige båndbredden. Senere, i 1983, tok ARPANET i bruk TCP/IP-modellen. Derfra var veien kort til Internett slik vi kjenner det i dag.

b)
Gjør først om src port, dest port, og size til binær, 16 bits presisjon:

$$
\begin{aligned}
2114_{10} &= 2048 + 64 + 2 \\
&= 2^{11} + 2^{6} + 2^{1} \\
&= 0000\ 1000\ 0100\ 0010_{2}
\end{aligned}
$$

$$
\begin{aligned}
53_{10} &= 32 + 16 + 4 + 1 \\
&= 2^{5} + 2^{4} + 2^{2} + 2^{0} \\
&= 0000\ 0000\ 0011\ 0101_{2}
\end{aligned}
$$

$$
\begin{aligned}
12_{10} &= 8 + 4 \\
&= 2^{3} + 2^{2} \\
&= 0000\ 0000\ 0000\ 1100_{2}
\end{aligned}
$$

Legger så sammen alle verdier, i tillegg til selve meldingen:

```
    0000 1000 0100 0010
+   0000 0000 0011 0101
=   0000 1000 0111 0111

                111 1
    0000 1000 0111 0111
+   0000 0000 0000 1100
=   0000 1000 1000 0011

       1 1111       11
    0000 1000 1000 0011
+   1100 1111 1111 0101
=   1101 1000 0111 1000
```

```
    1 11       111
    1101 1000 0111 1000
+   1100 1111 0001 0000
= 1 1010 0111 1000 1000
```

Legger til overflow:

```
    1010 0111 1000 1000
+   0000 0000 0000 0001
=   1010 0111 1000 1001
```

Tar så enerkomplement, og får sjekksummen:

```
    0101 1000 0111 0110
```

c)
Da nettmasken binært inneholder kun enere i de første og andre oktett, og kun nullere i den fjerde, trenger jeg bare å konvertere tredje oktett til binær i både nettmasken og begge IP-adressene for å komme frem til svaret.

Nettmaske, tredje oktett:

$$
\begin{aligned}
248_{10} &= 128 + 64 + 32 + 16 + 8 \\
&= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 \\
&= 1111\ 1000_2
\end{aligned}
$$

Første IP-adresse, tredje oktett:

$$
\begin{aligned}
42_{10} &= 32 + 8 + 2 \\
&= 2^5 + 2^3 + 2^1 \\
&= 0010\ 1010_2
\end{aligned}
$$

Andre IP-adresse, tredje oktett:

$$
\begin{aligned}
47_{10} &= 32 + 8 + 4 + 2 + 1 \\
&= 2^5 + 2^3 + 2^2 + 2^1 + 2^0 \\
&= 0010\ 1111_2
\end{aligned}
$$

Legger så tredje oktett av nettmasken på tredje oktett av begge IP-adresser:

```
  1111 1000
& 0010 1010
= 0010 1000

  1111 1000
& 0010 1111
= 0010 1000
```

Konverterer tallet til desimal:

$$2^5 + 2^3 = 40$$

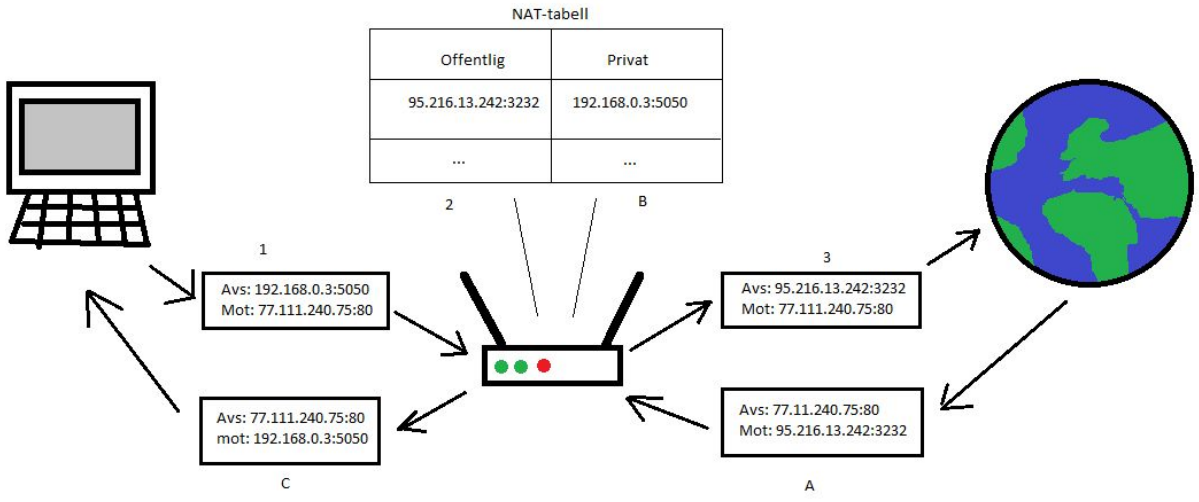
Setter det sammen med resten av nettmasken og IP-adressene; tar en snarvei her, i og med at jeg vet at 255 slipper gjennom hele verdien, mens 0 ikke slipper gjennom noe (mellomrom mellom oktetter lagt til for lesbarhet):

```
  255. 255. 248.  0
&  10. 112.  42. 29
=  10. 112.  40.  0

  255. 255. 248.  0
&  10.  21.  47.  2
=  10.  21.  40.  0
```

Da de to svarene ble ulike, må maskinene ligge på ulike IP-nett. De to maskinene kan kommunisere og sende hverandre data, men ikke direkte. De må gjøre det via en router, da de ligger på ulike subnett.

d)
NAT (Network Address Translation) er teknikken som brukes for å oversette mellom private og offentlige IPv4-adresser. Offentlige adresser er de som gjør det mulig å koble to maskiner sammen over Internett, mens private kun brukes internt i et LAN (Local Area Network).
Det er rutere som gir enheter på nettverket tilkoblet den nevnte ruteren en privat IP-adresse. Den samme private adressen kan brukes igjen på flere maskiner, såfremt de ligger på forskjellige LAN. Dette løser problemet med det begrensede antallet mulige IPv4-adresser. Uten dette måtte hver enhet hatt en dedikert offentlig IP-adresse, og per i dag har vi for mange mobiltelefoner, nettbrett, datamaskiner, spillkonsoller, smarte kjøleskap, o.l. i forhold til antall offentlige adresser tilgjengelig. Det er internettleverandører (ISP) som deler ut offentlige adresser til organisasjoner og husstander.



Fra privat til offentlig:

1. En maskin på et LAN sender en datapakke med sin private IP-adresse (med portnummer) som avsender.
2. Ruteren tar imot pakken, og endrer avsenderadressen til den offentlige IP-adressen, etter hva som står i NAT-tabellen. Om nødvendig, byttes også port. Tabellen blir oppdatert.
3. Mottaker får datapakken videresendt fra ruteren, nå med offentlig IP-adresse som avsender

Fra offentlig til privat:

A. En avsender fra Internett sender en datapakke adressert til en offentlig IP-adresse, med portnummer.
B. Ruteren tar imot pakken og sjekker den opp mot NAT-tabellen, og endrer mottakeradressen til den private IP-adressen. Tabellen oppdateres.
C. Maskinen på LAN'et får pakken sendt fra ruteren, nå med sin private IP-adresse som mottaksadresse.