Problem 1)

a. Host A determines inspecting the 8-bit protocol field in an IPv4 header or the next header field in an IPv6 header, which contains a numerical value indicating the appropriate transport layer protocol. I believe it is 6 for TCP and 17 for UDP.
b. Yes, a host can have multiple IPs through multiple networks like WiFi, Ethernet, etc. that each has its own unique IP address. Even a server can have multiple virtual interfaces with different IPs.
c. Session Traversal Utilities for NAT discovers public IPs and ports while Traversal Using Relays around NAT relays traffic through public servers if you can't connect peer-to-peer. For example if you had public IP A and your friend had public IP B, a STUN protocol can be used to discover the IPs and ports.
d. It wouldn't be needed as NAT existed in case of an IP address shortage under IPv4. With IPv6 we go from 2^{32} IPs to 2^{128} IP addresses which significantly reduces the need for NAT. Some firms might still prefer it though for security reasons.

Problem 2)

a. There are 8 subnets: 223.1.1.0, 223.1.2.0, 223.1.3.0, 223.1.4.0, 223.1.7.0, 223.1.8.0, 223.1.9.0, 223.1.10.0
b. DHCP discover (src: 0.0.0.0, dest: 255.255.255.255)
   DHCP offer (src: server IP, dest: 255.255.255.255)
   DHCP request (src: 0.0.0.0, dest: 255.255.255.255)
   DHCP ACK (src: server IP, dest: 255.255.255.255)
c. The main issue is that PC0 is now in the 223.1.10.0 subnet but has an IP from 223.1.2.0. This can lead to PC0 attempting to send data through a gateway that is no longer available. DHCP solves this problem by auto assigning PC0 a correct IP for its new subnet.

Problem 3)

If subnet 1 needs to hold 100 interfaces, subnet 2 needs to hold at least 50 interfaces, and subnet 3 needs to hold 25, we have 7, 6, and 5 host bits respectively. The subnet masks are (32 bits - 7 host bits =) 25, (32 bits - 6 host bits =) 26, and (32 bits - 5 host bits =) 27 respectively too. This means we have to allow $2^7 = 128$ spaces for subnet 1, $2^6 = 64$ spaces for subnet 2, and $2^5 = 32$ spaces for subnet 3.

NOTE: NO ADDRESSES CAN OVERLAP HERE

NOTE: we technically need x + 2 address spaces where x is the number of interfaces needed to support to include for broadcasting and network IP (but out of scope for this question but important for minimum address space)

S1: 1 _ _ _ _ _ _ _ => allows 128 distinct addresses for our requirement of 100, must flip first bit to 1 to prevent any overlap
S2: 0 1 _ _ _ _ _ _ => allows for 64 unique address spaces for our requirement of 50, 2nd most MSB has to be 1 to not coincide with S3.
S3: 0 0 _ _ _ _ _ _ => allows for 64 unique address spaces for our requirement of 25.

<u>Final Answers</u>:
Subnet 1: 10.0.0.128/25 (starts at 1 at MSB)
Subnet 2: 10.0.0.64/26 (starts at 1 at 2nd most MSB)
Subnet 3: 10.0.0.0/27 (starts at 0)

Problem 4)

a. ASSUME that we do not include the example's initial message

<u>IP: port within private network</u>	**<u>IP: port outside private network</u>**
`10.0.0.6:5000`	`131.179.176.1:8000`
`10.0.0.10:6000`	`131.179.176.1:8001`
`10.0.1.101:6001`	`131.179.176.1:8002`
`10.0.0.7:7000`	`131.179.176.1:8003`

For message 1: 10.0.0.6:5000 sends a message to 172.217.11.78:80
    We need to make a new entry because a packet is being sent from the private network. IP port outside private network starts at the default port 8000.
For message 2: 10.0.0.10:6000 sends a message to 204.79.197.200:80
    We make new entry for same reason as message 1
For message 3: 10.0.1.101:6001 sends a message to 206.190.36.45:80
    Make new entry for same reason as message 2
For message 4: 10.0.0.10:6000 sends a message to 204.79.197.200:80
    We don't need to make a new entry because 10.0.0.10:6000 already sent a message earlier (however if it was a different port number or IP we would need to make a new entry). Destination address doesn't matter for the table at all.
For message 5: 10.0.1.101:6001 sends a message to 172.217.11.78:80
    Already repeated, we do not need another entry
For message 6: 10.0.0.7:7000 sends a message to 63.245.215.20:80
    Requires new entry
For message 7: 204.79.197.200:80 sends a message to 131.179.176.1:8002
    Packet is coming from a port outside of the private network, it matches the IP for 10.0.1.101:6001 entry so we are fine. However if the destination IP doesn't match the IP: port outside the private network column at all we just drop it.

For message 8: 204.79.197.200:80 sends a message to 131.179.176.1:8003

Packet is coming from port outside of the private network, matches the IP for 10.0.0.7:7000

b.

(1) 10.0.0.6:5000 sends a message to 172.217.11.78:80:

Message Rcvd from Host: `MSG <10.0.0.6:5000, 172.217.11.78:80>`

Message Sent from Router: `MSG <131.179.176.1:8000, 172.217.11.78:80>`

(2) 10.0.0.10:6000 sends a message to 204.79.197.200:80

Message Rcvd from Host: `MSG <10.0.0.10:6000, 204.79.197.200:80>`

Message Sent from Router: `MSG <131.179.176.1:8001, 204.79.197.200:80>`

Problem 5)

Step by step implementation (highlight means current node):

Visited Set: {t}

Row	Cost
t:	0
u:	2
v:	4
w:	∞
x:	∞
y:	7
z:	∞

Visited Set: {t, u}

Row	Cost
t:	0
u:	2
v:	4 = min(4, 2 + 3)
w:	5 = 2 + 3
x:	∞

y:	7
z:	∞

Visited Set: {t, u, v}

Row	Cost
t:	0
u:	2 = min(2, 4 + 3)
v:	4
w:	5 = min(5, 4 + 4)
x:	7 = 3 + 4
y:	7 = min(7, 4 + 8)
z:	∞

Visited Set: {t, u, v, w}

Row	Cost
t:	0
u:	2 = min(2, 5 + 3)
v:	4 = min(4, 5 + 4)
w:	5
x:	7 = min(7, 5 + 6)
y:	7
z:	∞

Visited Set: {t, u, v, w, x}

Row	Cost
t:	0
u:	2

v:	4 = min(4, 7 + 3)
w:	5 = min(5, 7 + 6)
x:	7
y:	7 = min(7, 7 + 7)
z:	15 = 7 + 8

Visited Set: {t, u, v, w, x, y}

Row	**Cost**
t:	0
u:	2
v:	4 = min(4, 7 + 8)
w:	5
x:	7 = min(7, 7 + 6)
y:	7
z:	15 = min(15, 7 + 12)

Visited Set: {t, u, v, w, x, y, z}

Row	**Cost**
t:	0
u:	2
v:	4
w:	5
x:	7 = min(7, 15 + 8)
y:	7 = min(7, 15 + 12)
z:	15

Problem 6)

a. The initial distance from y => x is 4, distance from z => x is set to 50, but we have the alternatives z => y => x and z => w => y => x which is 7 and 6 respectively, getting us z => x is 6. w => x would be w => y => x and w => z => x which is 5 and 51 respectively, so w => x would be 5. In conclusion,
   w => x is 5
   y => x is 4
   z => x is 6
b. If y => x gets set to 60, w and z will still think that y has a cost of 4 to x so w thinks it can reach x via y in 1 + 60 = 61 while z thinks it can get to x in 1 + 1 + 60 = 62. But then each router thinks another has a shorter path leading to count to infinity problem, thus not being able to resolve the problem within 5 iterations.
c. It can once the router realizes that there is no low-cost path to x anymore and instead will stabilize to y => x being 60 units. This process can take many steps because count-to-infinity will take an insane amount of time to converge.