Computer Systems (2023-24)

Feedback on Quiz # 3 (Summative)

The **Quiz # 3** was composed of **10 questions**, which were randomly selected from a Question Bank of 30 questions. Answers and feedback comments for all of the questions are given below:

Q1.1	Which of the following statements are true?
(a)	The first DNS server accessed is always the root DNS server
(b)	A web browser uses the IMAP protocol to send emails
(c)	The HTTP protocol can only be used for the retrieval of html pages
(d)	**A persistent HTTP connection is more suitable for asking a server for an HTML page that references a lot of other objects on the same server**
(e)	Only the IP address of the server is needed to access the application that runs on it, e.g. to access a web server
(f)	**End User Network applications run on the network edge**
Feedback: a) The first DNS server to be accessed is usually the local ISP server b) Web-based email clients use HTTP protocol for sending emails c) The HTTP protocol can be used to retrieve many kinds of resources, not only html pages. For example, content-type can be video/mp4 d) A persistent HTTP connection allows the reuse of the same TCP connection which saves time by not establishing a new TCP connection on each request. It is beneficial because html pages typically contain a lot of style files and scripts. e) To access a web server the user application needs both the IP address and the port number on the machine where the server is running. f) Network applications run on network edge devices (hosts/end-systems)	

Q1.2	Which of the following statements are true?
(a)	The first DNS server accessed is always the root DNS server
(b)	The HTTP protocol can only be used for the retrieval of html pages
(c)	A non-persistent HTTP connection is more suitable for asking a server for an HTML page that references a lot of other objects on the same server
(d)	**The IP address and the port number of the server are needed to access the application that runs on it, e.g. to access a web server**
(e)	**End User Network applications run on the network edge**
(f)	A web browser uses the IMAP protocol to send emails
Feedback: a) The first DNS server accessed is usually the local ISP server	

b) The HTTP protocol can be used to retrieve many kinds of resources, not only html pages. For example, content-type can be video/mp4
c) A persistent HTTP connection allows the reuse of the same TCP connection which saves time by not establishing a new TCP connection on each request. Hence, using a non-persistent HTTP connection will incur an overhead by creating a new TCP connection every time a new object is accessed and therefore is less suitable.
d) To access a web server the user application needs both the IP address and the port number on the machine where the server is running.
e) Network applications run on network edge devices (hosts/end-systems)
f) Web-based email clients use HTTP protocol for sending emails

Q1.3	Which of the following statements are true?
(a)	End User Network applications run on the network core
(b)	**A web browser uses the HTTP protocol to send emails**
(c)	Only the IP address of the server is needed to access the application that runs on it, e.g. to access a web server
(d)	The first DNS server accessed is always the root DNS server
(e)	**A persistent HTTP connection is more suitable for asking a server for an HTML page that references a lot of other objects on the same server**
(f)	The HTTP protocol can only be used for the retrieval of html pages

Feedback:

a) Network applications run on network edge devices (hosts/end-systems)
b) Web-based email clients use HTTP protocol for sending emails
c) To access a web server the user application needs both the IP address and the port number on the machine where the server is running.
d) The first DNS server to access is usually the local ISP server
e) A persistent HTTP connection allows the reuse of the same TCP connection which saves time by not establishing a new TCP connection on each request. Hence, using a non-persistent HTTP connection will incur an overhead by creating a new TCP connection every time a new object is accessed and therefore is less suitable.
f) The HTTP protocol can be used to retrieve many kinds of resources, not only html pages. For example, content-type can be video/mp4

Q2.1	In the **SYN** message of the TCP connection establishing handshake, the client sends a 1-byte TCP segment to the server. That segment has Sequence number = 482 and SYNbit = 1. Which of the following TCP headers are correct for the corresponding **SYNACK** message from the server to the client?
(a)	**Sequence number = 500, Acknowledgement number = 483, SYNbit = 1, ACKbit = 1**
(b)	**Sequence number = 483, Acknowledgement number = 483, SYNbit = 1, ACKbit = 1**
(c)	Sequence number = 483, Acknowledgement number = 500, SYNbit = 1, ACKbit = 1
(d)	Sequence number = 500, Acknowledgement number = 483, SYNbit = 1, ACKbit = 0
(e)	Sequence number = 483, Acknowledgement number = 483, SYNbit = 1, ACKbit = 0
(f)	Sequence number = 483, Acknowledgement number = 500, SYNbit = 1, ACKbit = 0

Feedback:

In a SYNACK message, both the SYN and ACK bits must be 1, which discounts options (d), (e) and (f).
The acknowledgement number must be one plus the sequence number of the SYN message, which discounts option (c).
The sequence number of the SYNACK message is irrelevant, so both (a) and (b) are possible TCP headers for the message.

Q2.2	In the **SYNACK** message of the TCP connection establishing handshake, the server sends a 1-byte TCP segment to the client. That segment has Sequence number = 900, SYN/ACKbits = 1. Which of the following TCP headers are correct for the corresponding **ACK** message from the client to the server?
(a)	**Sequence number = 901, Acknowledgement number = 901, SYNbit = 0, ACKbit = 1**
(b)	**Sequence number = 500, Acknowledgement number = 901, SYNbit = 0, ACKbit = 1**
(c)	Sequence number = 901, Acknowledgement number = 483, SYNbit = 0, ACKbit = 1
(d)	Sequence number = 483, Acknowledgement number = 901, SYNbit = 1, ACKbit = 1
(e)	Sequence number = 901, Acknowledgement number = 901, SYNbit = 1, ACKbit = 1
(f)	Sequence number = 901, Acknowledgement number = 483, SYNbit = 1, ACKbit = 1

Feedback:

In an ACK message, the ACK bit must be 1 and the SYN bit must be 0, which discounts options (d), (e) and (f).
The acknowledgement number must be one plus the sequence number of the SYNACK message, which discounts option (c).
The sequence number of the ACK message is irrelevant, so both (a) and (b) are possible TCP headers for the message.

Q2.3	In the first **FIN** message of the TCP connection closing handshake, the client sends a 1-byte TCP segment to the server. That segment has Sequence number = 356 and FINbit = 1. Which of the following TCP headers are correct for the corresponding **ACK** message from the server to the client?
(a)	**Sequence number = 357, Acknowledgement number = 357, FINbit = 0, ACKbit = 1**
(b)	**Sequence number = 901, Acknowledgement number = 357, FINbit = 0, ACKbit = 1**
(c)	Sequence number = 357, Acknowledgement number = 901, FINbit = 0, ACKbit = 1
(d)	Sequence number = 357, Acknowledgement number = 901, FINbit = 1, ACKbit = 1
(e)	Sequence number = 901, Acknowledgement number = 901, FINbit = 1, ACKbit = 1
(f)	Sequence number = 901, Acknowledgement number = 357, FINbit = 1, ACKbit = 1

Feedback:

In an ACK message, the ACK bit must be 1 and the FIN bit must be 0, which discounts options (d), (e) and (f).

The acknowledgement number must be one plus the sequence number of the FIN message, which discounts option (c).

The sequence number of the ACK message is irrelevant, so both (a) and (b) are possible TCP headers for the message.

Q3.1	A UDP header plus its data is represented in **hexadecimal** as 0xF16C71A8718C. What will the UDP checksum be for this datagram?
(a)	**0010 1011 0101 1110**
(b)	0010 1011 0101 1111
(c)	0001 0101 1010 1111
(d)	0001 0101 1010 1110

Feedback:

First, we convert the hex into binary: F16C71A8718C becomes

```
111100010110110001110001101010000111000110001100
```

Then, we break this into 16-bit chunks:

```
1111 0001 0110 1100
0111 0001 1010 1000
0111 0001 1000 1100
```

These are added together, with carry bits added back in:

```
  1111 0001 0110 1100
+ 0111 0001 1010 1000
=10110 0011 0001 0100
+                   1
= 0110 0011 0001 0101
+ 0111 0001 1000 1100
= 1101 0100 1010 0001
```

Each digit is flipped to make the checksum: **0010 1011 0101 1110**

Q3.2	A UDP header plus its data is represented in **hexadecimal** as 0xCD00932ABB80. What will the UDP checksum be for this datagram?
(a)	**1110 0100 0101 0011**
(b)	1111 0100 0101 0100
(c)	0000 1011 1010 1010
(d)	0000 1011 1010 1011

Feedback:
First, we convert the hex into binary: CD00932ABB80 becomes
110011010000000010010011001010101011101110000000

Then, we break this into 16-bit chunks:

```
1100 1101 0000 0000
1001 0011 0010 1010
1011 1011 1000 0000
```

These are added together, with carry bits added back in:

```
  1100 1101 0000 0000
+ 1001 0011 0010 1010
=10110 0000 0010 1010
+                   1
= 0110 0000 0010 1011
+ 1011 1011 1000 0000
=10001 1011 1010 1011
+                   1
= 0001 1011 1010 1100
```

Each digit is flipped to make the checksum: **1110 0100 0101 0011**

Q3.3	A UDP header plus its data is represented in **hexadecimal** as 0x8923CA0832DF. What will the UDP checksum be for this datagram?
(a)	**0111 1001 1111 0100**
(b)	0111 1001 1111 0011
(c)	1000 0110 0000 1100
(d)	1000 0110 0000 1011

Feedback:
First, we convert the hex into binary: 8923CA0832DF becomes
100010010010001111001010000010000011001011011111

Then, we break this into 16-bit chunks:
`1000 1001 0010 0011`
`1100 1010 0000 1000`
`0011 0010 1101 1111`
 	
These are added together, with carry bits added back in:
`  1000 1001 0010 0011`
`+ 1100 1010 0000 1000`
`=`**`1`**`0101 0011 0010 1011`
`+                   `**`1`**
`= 0101 0011 0010 1100`
`+ 0011 0010 1101 1111`
`= 1000 0110 0000 1011`
 	
Each digit is flipped to make the checksum: **`0111 1001 1111 0100`**

Q4.1	Suppose that you have intercepted the following HTTP packet
	
	<u>`00 00 00 00 45 00 01 44 00 00 40 00 40 06 00 00`</u>
	<u>`7f 00 00 01 7f 00 00 01 46 50 fa 8c 91 83 a5 04`</u>
	<u>`66 6f 1c 98 80 18 18 ea ff 38 00 00 01 01 08 0a`</u>
	<u>`72 1e a3 42 93 de 97 c1`</u> `48 54 54 50 2f 31 2e 31`
	`20 32 30 30 20 4f 4b 0d 0a 58 2d 50 6f 77 65 72`
	`65 64 2d 42 79 3a 20 45 78 70 72 65 73 73 0d 0a`
	`41 63 63 65 73 73 2d 43 6f 6e 74 72 6f 6c 2d 41`
	`6c 6c 6f 77 2d 4f 72 69 67 69 6e 3a 20 2a 0d 0a`
	`43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65`
	`78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 73 65 74`
	`3d 75 74 66 2d 38 0d 0a 43 6f 6e 74 65 6e 74 2d`
	`4c 65 6e 67 74 68 3a 20 31 33 0d 0a 45 54 61 67`
	`3a 20 57 2f 22 64 2d 76 5a 44 65 71 34 62 51 35`
	`65 58 7a 75 6b 42 42 6f 6c 35 66 79 71 30 53 78`
	`33 30 22 0d 0a 44 61 74 65 3a 20 53 61 74 2c 20`
	`32 35 20 4e 6f 76 20 32 30 32 33 20 31 31 3a 31`
	`33 3a 32 30 20 47 4d 54 0d 0a 43 6f 6e 6e 65 63`
	`74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65`
	`0d 0a 4b 65 65 70 2d 41 6c 69 76 65 3a 20 74 69`
	`6d 65 6f 75 74 3d 35 0d 0a 0d 0a 57 6f 72 6c 64`
	`20 48 65 6c 6c 6f 21 0a`

The first 56 bytes (bold and underlined) correspond to Link Layer, IP and TCP protocols in which the HTTP message is wrapped. Does this intercepted packet correspond to an HTTP request or response? How long is the body part of the HTTP packet in bytes?

Remember that HTTP is a text protocol. Thus, each pair of hexadecimal numbers corresponds to a character. Use the following table for decoding. \r is a carriage return and \n is a line feed.

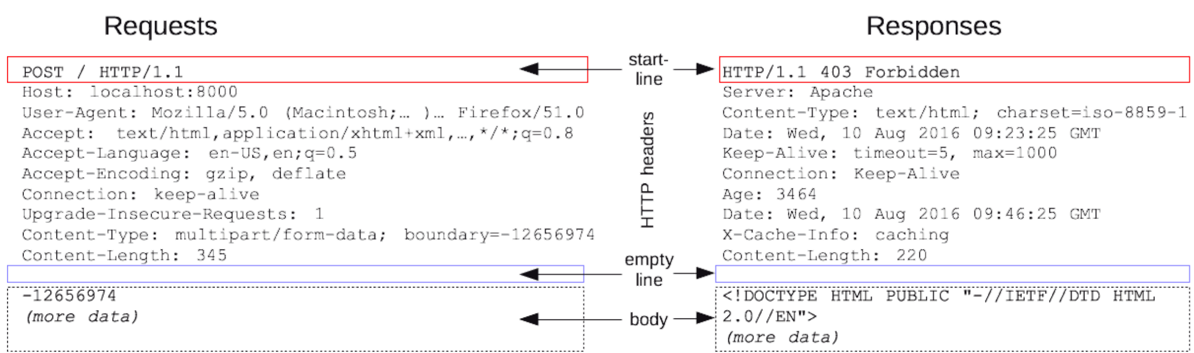
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

(a)	**HTTP Response, 13**
(b)	HTTP Response, 26
(c)	HTTP Request, 13
(d)	HTTP Request, 26

Feedback:

Recall the structure of HTTP message



and recall that the header is encoded in ASCII which means that every byte in the header corresponds to a symbol in the ASCII table. To determine whether a message is a request or a response, we need to have a look at the first few bytes of the HTTP packet: if the packet starts with POST, GET etc. then this is a request, otherwise the packet should start with HTTP and be a response. Consider the first few bytes after skipping the initial 56 bytes: `48 54 54 50 2f 31 2e` these correspond to HTTP/1.1 (48 is the ASCII code for H, 54 is the ASCII code for T and so on) and hence this is a response. The HTTP protocol does not impose any particular encoding for the body of the HTTP packet, but we are merely interested in the number of bytes in the body. The body of any HTTP packet starts right after the first empty line, i.e. just after the occurrence of \r\n\r\n which is encoded as `0d 0a 0d 0a` (one `0d 0a` corresponds to \r\n for the last line of the header and the other `0d 0a` is for the empty string). By locating this occurrence of `0d 0a 0d 0a` and counting the remaining bytes we get 13 bytes. Another approach would be using hex to ASCII converter, e.g. https://www.rapidtables.com/convert/number/hex-to-ascii.html which would display the message as plain text.

Q4.2	Suppose you have intercepted the following HTTP packet **c0 d7 aa 95 ad 43 d0 88 0c 7e 4d 28 08 00 45 00** **00 fe 00 00 40 00 40 06 8e dc c0 a8 01 bf 36 57** **b2 5f d6 c4 00 50 9d 55 8a cb 6c 9b 89 51 80 18** **08 16 17 23 00 00 01 01 08 0a 9f 29 ce 4b d9 d4** **0a 2e** 50 4f 53 54 20 2f 70 6f 73 74 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 68 74 74 70 62 69 6e 2e 6f 72 67 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 63 75 72 6c 2f 37 2e 38 34 2e 30 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 6a 73 6f 6e 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 37 30 0d 0a 0d 0a 7b 22 74 65 78 74 22 20 3a 20 22 54 68 69 73 20 69 73 20 61 20 73 61 6d 70 6c 65 20 50 4f 53 54 20 72 65 71 75 65 73 74 20 66 6f 72 20 43 6f 6d 70 75 74 65 72 20 53 79 73 74 65 6d 73 20 6d 6f 64 75 6c 65 22 7d

The first 66 bytes (bold and underlined) correspond to Link Layer, IP and TCP protocols in which the HTTP message is wrapped. Does this intercepted packet correspond to an HTTP request or response? How long is the body part of the HTTP packet in bytes?

Remember that HTTP is a text protocol. Thus, each pair of hexadecimal numbers corresponds to a character. Use the following table for decoding. \r is a carriage return and \n is a line feed.

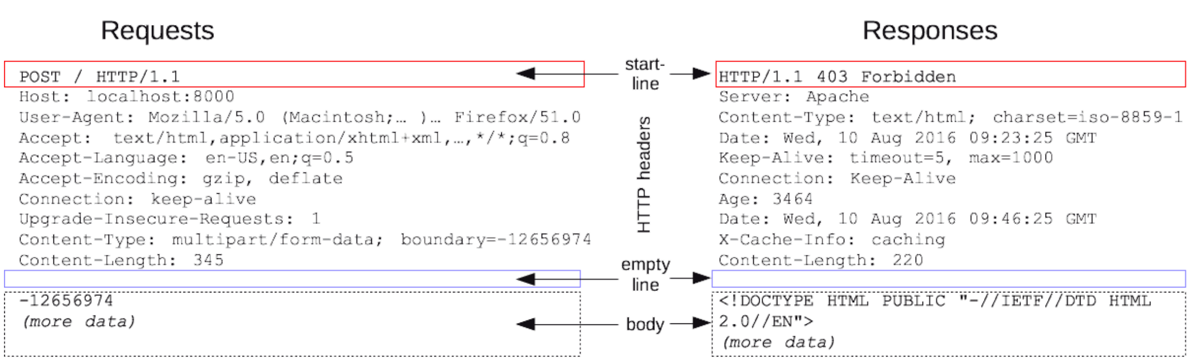
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

(a)	HTTP Response, 70
(b)	**HTTP Request, 70**
(c)	HTTP Response, 140
(d)	HTTP Request, 140

Feedback:

Recall the structure of HTTP message



and recall that the header is encoded in ASCII which means that every byte in the header corresponds to a symbol in the ASCII table. To determine whether a message is a request or a response, we need to have a look at the first few bytes of the HTTP packet: if the packet starts with POST, GET etc. then this is a request, otherwise the packet should start with HTTP and be a response.

Consider the first few bytes after skipping the initial 66 bytes: `50 4f 53 54` these correspond to POST (50 is the ASCII code for P, 4f is the ASCII code for O and so on) and hence this is a HTTP request. The HTTP protocol does not impose any particular encoding for the body of the HTTP packet, but we are merely interested in the number of bytes in the body. The body of any HTTP packet starts right after the first empty line, i.e. just after the occurrence of \r\n\r\n which is encoded as `0d 0a 0d 0a` (one `0d 0a` corresponds to \r\n for the last line of the header and the other `0d 0a` is for the empty string). By locating this occurrence of `0d 0a 0d 0a` and counting the remaining bytes we get 70 bytes.

Another approach would be using hex to ASCII converter, e.g. https://www.rapidtables.com/convert/number/hex-to-ascii.html which would display the message as plain text.

Q4.3 Suppose you have intercepted the following HTTP packet

```
c0 d7 aa 95 ad 43 d0 88 0c 7e 4d 28 86 dd 60 0e
00 00 00 f1 06 40 2a 00 23 c7 fe 00 53 01 21 ae
a6 63 53 55 0a 01 26 06 47 00 00 20 00 00 00 00
00 00 ac 43 48 f9 d9 19 00 50 07 71 89 57 c2 c4
7e bd 80 18 08 10 8e 00 00 00 01 01 08 0a 0a 3e
24 42 79 e6 bb fa 47 45 54 20 2f 65 63 68 6f 2f
70 6f 73 74 2f 6a 73 6f 6e 20 48 54 54 50 2f 31
2e 31 0d 0a 48 6f 73 74 3a 20 72 65 71 62 69 6e
2e 63 6f 6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74
3a 20 63 75 72 6c 2f 37 2e 38 34 2e 30 0d 0a 41
63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 43 6f 6e 74
65 6e 74 2d 54 79 70 65 3a 20 61 70 70 6c 69 63
61 74 69 6f 6e 2f 6a 73 6f 6e 0d 0a 43 6f 6e 74
65 6e 74 2d 4c 65 6e 67 74 68 3a 20 36 39 0d 0a
0d 0a 7b 22 74 65 78 74 22 20 3a 20 22 54 68 69
73 20 69 73 20 61 20 73 61 6d 70 6c 65 20 47 45
54 20 72 65 71 75 65 73 74 20 66 6f 72 20 43 6f
6d 70 75 74 65 72 20 53 79 73 74 65 6d 73 20 6d
6f 64 75 6c 65 22 7d
```

The first 86 bytes (bold and underlined) correspond to Link Layer, IP and TCP protocols in which the HTTP message is wrapped. Does this intercepted packet correspond to an HTTP request or response? How long is the body part of the HTTP packet in bytes?

Remember that HTTP is a text protocol. Thus, each pair of hexadecimal numbers corresponds to a character. Use the following table for decoding. \r is a carriage return and \n is a line feed.

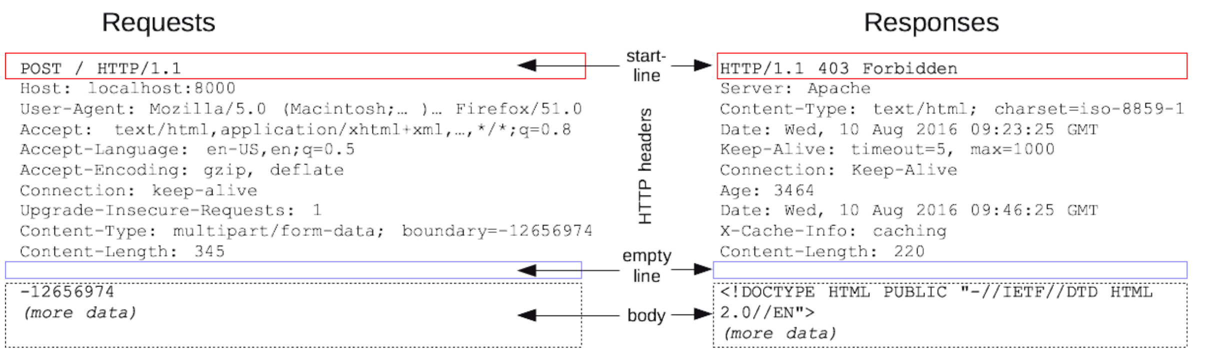
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

(a)	HTTP Response, 69
(b)	**HTTP Request, 69**
(c)	HTTP Response, 138
(d)	HTTP Request, 138

Feedback:

Recall the structure of HTTP message



and recall that the header is encoded in ASCII which means that every byte in the header corresponds to a symbol in the ASCII table. To determine whether a message is a request or a response, we need to have a look at the first few bytes of the HTTP packet: if the packet starts with POST, GET etc. then this is a request, otherwise the packet should start with HTTP and be a response.

Consider the first few bytes after skipping the initial 86 bytes: `47 45 54` these correspond to GET (47 is the ASCII code for G, 45 is the ASCII code for E and so on) and hence this is a request. The HTTP protocol does not impose any particular encoding for the body of the HTTP packet, but we are merely interested in the number of bytes in the body. The body of any HTTP packet starts right after the first empty line, i.e. just after the occurrence of \r\n\r\n which is encoded as `0d 0a 0d 0a` (one `0d 0a` corresponds to \r\n for the last line of the header and the other `0d 0a` is for the empty string). By locating this occurrence of `0d 0a 0d 0a` and counting the remaining bytes we get 69 bytes.

Another approach would be using hex to ASCII converter, e.g. https://www.rapidtables.com/convert/number/hex-to-ascii.html which would display the message as plain text.

Q5.1	Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	7 ms
P2	3 ms	3 ms
P3	5 ms	10 ms

What is the average turnaround time assuming **SRTF** CPU scheduling policy?

(a)	**9.33 ms**
(b)	8.67 ms
(c)	10.00 ms
(d)	6.67 ms

Feedback:

For the given set of processes, here is the SRTF schedule:

P_1	P_1	P_1	P_2	P_2	P_2	P_1	P_1	P_1	P_1	P_3	P_3	P_3	P_3	P_3	P_3	P_3	P_3	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P1 = (Process Completion – Arrival Time) = (10 – 0) = 10 ms

Turnaround Time for P2 = (Process Completion – Arrival Time) = (6 – 3) = 3 ms

Turnaround Time for P3 = (Process Completion – Arrival Time) = (20 – 5) = 15 ms

Average Turnaround Time = (10 + 3 + 15) / 3 = 9.33 ms.

Q5.2	Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	6 ms
P2	4 ms	10 ms
P3	7 ms	4 ms

What is the average turnaround time assuming **SRTF** CPU scheduling policy?

(a)	9.33 ms
(b)	**8.67 ms**
(c)	10.00 ms
(d)	6.67 ms

Feedback:

For the given set of processes, here is the SRTF schedule:

P_1	P_1	P_1	P_1	P_1	P_1	P_2	P_3	P_3	P_3	P_3	P_2	P_2	P_2	P_2	P_2	P_2	P_2	P_2	P_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P1 = (Process Completion – Arrival Time) = (6 – 0) = 6 ms

Turnaround Time for P2 = (Process Completion – Arrival Time) = (20 – 4) = 16 ms

Turnaround Time for P3 = (Process Completion – Arrival Time) = (11 – 7) = 4 ms

Average Turnaround Time = (6 + 16 + 4) / 3 = 8.67 ms.

Q5.3 Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	10 ms
P2	5 ms	4 ms
P3	8 ms	6 ms

What is the average turnaround time assuming **SRTF** CPU scheduling policy?

(a)	9.33 ms
(b)	8.67 ms
(c)	**10.00 ms**
(d)	6.67 ms

Feedback:

For the given set of processes, here is the SRTF schedule:

P_1	P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_1	P_1	P_1	P_1	P_1	P_3	P_3	P_3	P_3	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P1 = (Process Completion – Arrival Time) = (14 – 0) = 14 ms

Turnaround Time for P2 = (Process Completion – Arrival Time) = (9 – 5) = 4 ms

Turnaround Time for P3 = (Process Completion – Arrival Time) = (20 – 8) = 12 ms

Average Turnaround Time = (14 + 4 + 12) / 3 = 10.00 ms.

Q6.1	Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has four links, of rates R1 = 5 Mbps, R2 = 4 Mbps, R3 = 10 Mbps and R4 = 20 Mbps. What will be the minimum end-to-end delay of transferring a video clip of 250 MegaBytes? (We assume that there is no store-and-forward delay on the routers connecting the links)
(a)	**524.29 seconds**
(b)	419.43 seconds
(c)	209.72 seconds
(d)	104.86 seconds

Feedback:
As there are four links, the bottle-neck link will be R2, which has 4 Mbps bandwidth. We can compute the end-to-end delay (ideal case i.e. minimum) as below:

Size of file = 250 MB = 250 * 1024 * 1024 bytes = 250 * 1024 * 1024 * 8 bits
Effective Bandwidth (Throughput) = 4 Mbps = 4 * 10^6 bps
End-to-end delay = (250 * 1024 * 1024 * 8) / (4 * 10^6) = 524.29 seconds

Q6.2	Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has four links, of rates R1 = 5 Mbps, R2 = 20 Mbps, R3 = 10 Mbps and R4 = 8 Mbps. What will be the minimum end-to-end delay of transferring a video clip of 370 MegaBytes? (We assume that there is no store-and-forward delay on the routers connecting the links)
(a)	**620.76 seconds**
(b)	155.19 seconds
(c)	310.38 seconds
(d)	387.97 seconds

Feedback:
As there are four links, the bottle-neck link will be R1, which has 5 Mbps bandwidth. We can compute the end-to-end delay (ideal case i.e. minimum) as below:

Size of file = 370 MB = 370 * 1024 * 1024 bytes = 370 * 1024 * 1024 * 8 bits
Effective Bandwidth (Throughput) = 5 Mbps = 5 * 10^6 bps
End-to-end delay = (370 * 1024 * 1024 * 8) / (5 * 10^6) = 620.76 seconds

Q6.3	Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has four links, of rates R1 = 12 Mbps, R2 = 15 Mbps, R3 = 8 Mbps and R4 = 25 Mbps. What will be the minimum end-to-end delay of transferring a video clip of 535 MegaBytes? (We assume that there is no store-and-forward delay on the routers connecting the links)
(a)	**560.99 seconds**
(b)	373.99 seconds
(c)	299.19 seconds
(d)	179.52 seconds

Feedback:

As there are four links, the bottle-neck link will be R3, which has 8 Mbps bandwidth. We can compute the end-to-end delay (ideal case i.e. minimum) as below:

Size of file = 535 MB = 535 * 1024 * 1024 bytes = 535 * 1024 * 1024 * 8 bits
Effective Bandwidth (Throughput) = 8 Mbps = 8 * 10^6 bps
End-to-end delay = (535 * 1024 * 1024 * 8) / (8 * 10^6) = 560.99 seconds

Q7.1	A subnet has *exactly* 512 addresses on its network (including the subnet and broadcast addresses). The IP address of *one* host machine on this subnet is **198.27.11.90**. What is the range of IP addresses in this subnet?
(a)	**198.27.10.0 - 198.27.11.255**
(b)	198.27.11.0 - 198.27.12.255
(c)	198.27.11.90 - 198.27.13.89
(d)	198.27.11.247 - 198.27.11.255

Feedback:

To have 512 addresses, we need log2(512) = 9 bits in the host part. So the subnet part has 32 - 9 = 23 bits. Therefore, the IP address of the host machine in CIDR format is 198.27.11.90/23.

198.27.11.90 in binary is 11000110.00011011.0000101 1.01011010, and the underlined bits refer to the subnet part. This means the first (subnet) address is 11000110.00011011.0000101 0.00000000 and the final (broadcast) address is 11000110.00011011.0000101 1.11111111.

Converting this back to dotted-decimal, the range is 198.27.10.0-198.27.11.255.

Q7.2	A subnet has *exactly* 8 addresses on its network (including the subnet and broadcast addresses). The IP address of *one* host machine on this subnet is **198.27.11.90**. What is the range of IP addresses in this subnet?
(a)	**198.27.11.88 - 198.27.11.95**
(b)	198.27.11.90 - 198.27.11.97
(c)	198.27.11.247 - 198.27.11.255
(d)	198.27.11.83 - 198.27.11.90
Feedback: To have 8 addresses, we need log2(8) = 3 bits in the host part. So the subnet part has 32 - 3 = 29 bits. Therefore, the IP address of the host machine in CIDR format is 198.27.11.90/29. 198.27.11.90 in binary is 11000110.00011011.00001011.01011010, and the underlined bits refer to the subnet part. This means the first (subnet) address is 11000110.00011011.00001011.01011000 and the final (broadcast) address is 11000110.00011011.00001011.01011111. Converting this back to dotted-decimal, the range is 198.27.11.88-198.27.11.95.	

Q7.3	A subnet has *exactly* 1024 addresses on its network (including the subnet and broadcast addresses). The IP address of *one* host machine on this subnet is **198.27.11.90**. What is the range of IP addresses in this subnet?
(a)	**198.27.8.0 - 198.27.11.255**
(b)	198.27.11.0 - 198.27.11.255
(c)	198.27.11.90 - 198.27.15.89
(d)	198.27.10.0 - 198.27.11.255
Feedback: To have 1024 addresses, we need log2(1024) = 10 bits in the host part. So the subnet part has 32 - 10 = 22 bits. Therefore, the IP address of the host machine in CIDR format is 198.27.11.90/22. 198.27.11.90 in binary is 11000110.00011011.00001011.01011010, and the underlined bits refer to the subnet part. This means the first (subnet) address is 11000110.00011011.00001000.00000000 and the final (broadcast) address is 11000110.00011011.00001011.11111111. Converting this back to dotted-decimal, the range is 198.27.8.0-198.27.11.255.	

Q8.1 Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	5 ms
P2	2 ms	8 ms
P3	3 ms	6 ms
P4	5 ms	2 ms

What is the average response time assuming **Round Robin** CPU scheduling policy with a Time Quantum (q = 4 ms). Also note that newly arriving processes are added to the tail of the ready queue.

(a)	**3.75 ms**
(b)	3.00 ms
(c)	5.25 ms
(d)	4.33 ms

Feedback:

For the given set of processes, here is the RR schedule:

P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3	P_1	P_4	P_4	P_2	P_2	P_2	P_2	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Response Time for P1 = (Process Execution Start – Arrival Time) = (0 – 0) = 0 ms
Response Time for P2 = (Process Execution Start – Arrival Time) = (4 – 2) = 2 ms
Response Time for P3 = (Process Execution Start – Arrival Time) = (8 – 3) = 5 ms
Response Time for P4 = (Process Execution Start – Arrival Time) = (13 – 5) = 8 ms
Average Response Time = (0 + 2 + 5 + 8) / 4 = 3.75 ms.

Q8.2 Consider the following set of processes:

Processes	**Arrival Time**	**Burst Time**
P1	0 ms	4 ms
P2	1 ms	9 ms
P3	2 ms	3 ms
P4	4 ms	5 ms

What is the average response time assuming **<u>Round Robin</u>** CPU scheduling policy with a Time Quantum (q = 3 ms). Also note that newly arriving processes are added to the tail of the ready queue.

(a)	3.75 ms
(b)	**3.00 ms**
(c)	5.25 ms
(d)	4.33 ms

Feedback:

For the given set of processes, here is the RR schedule:

P_1	P_1	P_1	P_2	P_2	P_2	P_3	P_3	P_3	P_1	P_4	P_4	P_4	P_2	P_2	P_2	P_4	P_4	P_2	P_2	P_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Response Time for P1 = (Process Execution Start – Arrival Time) = (0 – 0) = 0 ms
Response Time for P2 = (Process Execution Start – Arrival Time) = (3 – 1) = 2 ms
Response Time for P3 = (Process Execution Start – Arrival Time) = (6 – 2) = 4 ms
Response Time for P4 = (Process Execution Start – Arrival Time) = (10 – 4) = 6 ms
Average Response Time = (0 + 2 + 4 + 6) / 4 = 3.00 ms.

Q8.3	Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	8 ms
P2	2 ms	6 ms
P3	4 ms	5 ms
P4	6 ms	2 ms

What is the average response time assuming **Round Robin** CPU scheduling policy with a Time Quantum (q = 5 ms). Also note that newly arriving processes are added to the tail of the ready queue.

(a)	3.75 ms
(b)	3.00 ms
(c)	**5.25 ms**
(d)	4.33 ms

Feedback:

For the given set of processes, here is the RR schedule:

P_1	P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3	P_3	P_1	P_1	P_1	P_4	P_4	P_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Response Time for P1 = (Process Execution Start – Arrival Time) = (0 – 0) = 0 ms

Response Time for P2 = (Process Execution Start – Arrival Time) = (5 – 2) = 3 ms

Response Time for P3 = (Process Execution Start – Arrival Time) = (10 – 4) = 6 ms

Response Time for P4 = (Process Execution Start – Arrival Time) = (18 – 6) = 12 ms

Average Response Time = (0 + 3 + 6 + 12) / 4 = 5.25 ms.

Q9.1	Which of the following statements are true?
(a)	**Every thread within a process shares the same data space**
(b)	**Creating a new thread is faster than creating a new process**
(c)	**Every thread has its own copy of the program counter and registers**
(d)	Threads within a process must run on different processor cores
(e)	Threads can never execute concurrently
(f)	All threads within a process share the same stack

Feedback:

Threads exist within a process. Because each thread can run concurrently, each thread will have its own program counter, stack and registers. They will, though, share the same data space (of their process).

Creating a new thread is, therefore, much faster than creating a new process.

Threads within a process can be mapped to a single kernel thread but usually they are mapped to separate kernel threads which will be scheduled by the kernel. This means that they can run concurrently and in parallel if they are scheduled on different processor cores.

Threads share the same data space and so they can interact through this shared memory.

Since threads can run concurrently, the programmer has to manage the potential interactions between threads (e.g. through locks). This can lead to deadlocks.

Q9.2	Which of the following statements are true?
(a)	Whenever a new thread is created, all the data must be copied
(b)	Multiple threads within a process can never become deadlocked
(c)	All threads within a process must map to a single kernel thread
(d)	**Threads within a process can interact through shared memory**
(e)	**Every thread has its own program counter and registers**
(f)	**Every thread has its own program counter, registers and stack**

Feedback:

Threads exist within a process. Because each thread can run concurrently, each thread will have its own program counter, stack and registers. They will, though, share the same data space (of their process).

Creating a new thread is, therefore, much faster than creating a new process.

Threads within a process can be mapped to a single kernel thread but usually they are mapped to separate kernel threads which will be scheduled by the kernel. This means that they can run concurrently and in parallel if they are scheduled on different processor cores.

Threads share the same data space and so they can interact through this shared memory.

Since threads can run concurrently, the programmer has to manage the potential interactions between threads (e.g. through locks). This can lead to deadlocks.

Q9.3	Which of the following statements are true?
(a)	All threads within a process share the same program counter
(b)	Threads in different processes share the same data
(c)	**Creating a new thread is faster than creating a new process**
(d)	Threads within a process can never execute concurrently
(e)	**It is possible for deadlock to occur between two or more threads within a process**
(f)	**Every thread has its own stack**

Feedback:
Threads exist within a process. Because each thread can run concurrently, each thread will have its own program counter, stack and registers. They will, though, share the same data space (of their process).
Creating a new thread is, therefore, much faster than creating a new process.
Threads within a process can be mapped to a single kernel thread but usually they are mapped to separate kernel threads which will be scheduled by the kernel. This means that they can run concurrently and in parallel if they are scheduled on different processor cores.
Threads share the same data space and so they can interact through this shared memory.
Since threads can run concurrently, the programmer has to manage the potential interactions between threads (e.g. through locks). This can lead to deadlocks.

Q10.1	You have a program which takes 1000 seconds to run on a single processor core. 20% of this program execution is inherently sequential. The remaining 80% can be run in parallel. Which of the values below *best represents* how long will this program take to execute on a machine with 10 processor cores?
(a)	280 seconds
(b)	< 280 seconds
(c)	**> 280 seconds**
(d)	< 820 seconds
(e)	820 seconds
(f)	100 seconds

Feedback:
20% of the execution time is not parallelizable = 200 seconds
The remaining 80% (800 seconds) is parallelizable. If this is spread over 10 processors then, potentially, this could be completed in 80 seconds.
This would give a total execution time of 280 seconds (200+80).
In practice, overheads (memory access, cache limitations etc.) will mean that this theoretical maximum will not be reached. Therefore, it will take more than 280 seconds.
Amdahl's law is one way to describe this behaviour.

Q10.2	You have a program which takes 500 seconds to run on a single processor core. 10% of this program execution is inherently sequential. The remaining 90% can be run in parallel. Which of the values below *best represents* how long will this program take to execute on a machine with 20 processor cores?
(a)	25 seconds
(b)	22.5 seconds
(c)	72.5 seconds
(d)	< 72.5 seconds
(e)	**> 72.5 seconds**
(f)	< 22.5 seconds

Feedback:
10% of the execution time is not parallelizable = 50 seconds
The remaining 90% (450 seconds) is parallelizable. If this is spread over 20 processors then, potentially, this could be completed in 22.5 seconds.
This would give a total execution time of 72.5 seconds (50+22.5).
In practice, overheads (memory access, cache limitations etc.) will mean that this theoretical maximum will not be reached. Therefore it will take longer than 72.5 seconds.
Amdahl's law is one way to describe this behaviour.

Q10.3	You have a program which takes 10000 seconds to run on a single processor core. 30% of this program execution is inherently sequential. The remaining 70% can be run in parallel. Which of the values below *best represents* how long will this program take to execute on a machine with 5 processor cores?
(a)	2000 seconds
(b)	4400 seconds
(c)	**> 4400 seconds**
(d)	< 2000 seconds
(e)	> 1400 seconds
(f)	1400 seconds

Feedback:
30% of the execution time is not parallelizable = 3000 seconds
The remaining 70% (7000 seconds) is parallelizable. If this is spread over 5 processors then, potentially, this could be completed in 1400 seconds.
This would give a total execution time of 4400 seconds (3000+1400).
In practice, overheads (memory access, cache limitations etc.) will mean that this theoretical maximum will not be reached. Therefore, it will take longer than 4400 seconds
Amdahl's law is one way to describe this behaviour.