Q1

[3 marks] What is the major difference between multi-programmed and timesharing systems? Be sure to note the impact this difference has on scheduling.

multi-programmed systems are designed to maximise processor use, whereas timesharing systems are designed to minimise wait times. Multi-programmed systems are able to schedule other processes when a process becomes blocked. Timesharing systems improve on multi-user responsiveness by breaking processing into time slices; each process gets a limited amount of time on the cpu before a different process is scheduled.

Q2

Consider the following set of processes, with the length of the CPU-burst time given in Milliseconds:

process	burst	Priority
P1	6	3
P2	4	2
P3	1	4
P4	6	1
P5	2	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0. You may neglect the cost of context switching.

(a) [6 marks] Draw three Gantt charts illustrating the execution of these processes using SJF, RR (q = 2), and priority (a smaller priority number implies a higher priority) scheduling.

SJF	p3	p5	p2				p1					p4							
RR	p1		p2		p3	p4		p5		p1		p2		p4		p1		p4	
priority	p4						p2				p1						p3	p5	
time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

(b) [3 marks] What is the average turnaround time for each of the algorithms in question 2(a).

SJF: (1+7+13+19)/5 = 8
RR: (17+13+5+19+9)/5 = 12.6
Priority: (16+10+17+6+19)/5 = 13.6

(c) [3 marks] What is the average waiting time for each of the algorithms in question 2(a).

SJF: (7+3+0+13+1)/5 = 4.8
RR: (11+9+4+13+7)/5 = 8.8
Priority: (10+6+16+0+17)/5 = 9.8

Q3

(a) [2 Marks] Define external and internal fragmentation with reference to memory allocation.

External fragmentation: sections of memory that are available for allocation but are too small to hold any process; there may be enough external fragments to hold a process but they can't be used as they are not contiguous.

Internal fragmentation: sections of memory allocated to a process that the process doesn't need, they can't be allocated to another process and will go unused.

(b) [3 marks] Explain why the worst-fit memory allocation method may reduce external fragmentation, whereas the best-fit method may increase external fragmentation.

The best-fit method allocates the smallest available chunk of memory that fits the process, this leaves many small chunks of memory that no process can fit into. Worst-fit memory allocation selects the largest available chunk, thus the leftover fragments of memory are as large as possible minimising the number of unallocatable chunks.

(c) [4 Marks] Compare and contrast paging and segmentation with regards to fragmentation.

Paging breaks logical and physical memory into pages and frames of a fixed size, this allows processes to be allocated noncontiguous physical memory. Not being limited to contiguous memory avoids external fragmentation, however paging still suffers internal fragmentation when processes need a non-integer number of pages. Larger pages means greater internal fragmentation.
Segmentation is designed from a user perspective instead of a hardware perspective, it is intended to enable sharing between processes. Segments are named logical units with specific purposes. Segments can be any size, the segment table stores the length and base location of the segment. Because segments are arbitrary size they don't have internal fragmentation but will have problems with external fragments.
Paging and Segmentation can be used together to minimise both internal and external fragmentation. When used together each segment maps to a number of potentially noncontiguous pages.

Q4

Consider a computer system in which each memory page comprises of 1024 bytes of continuous memory in a 2^{16} memory address space. Each process uses only one page table. Assume that every entry in the page table is 2 bytes (1 byte = 8 bits).

(a) [2 marks] How much memory would we need to implement one page table?

2^16 / 1024 = 64 pages
64 * 2 = 128 bytes

(b) [4 marks] If we want to fully utilise the memory in the page holding the page table, what size pages would we need? If we can't do this - why not?

(2^16 / page size) * 2 = table size
page size = (2^16 *2) / table size
(2^16 / 361) * 2 = 363.08
(2^16 / 362) * 2 = 362.077
(2^16 / 363) * 2 = 361.08
There is no point where a whole number of bytes as the page size can map to an equal whole number of bytes as the page table.