

Feedback on Quiz # 2 (Summative)

The **Quiz # 2** 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	Select all the following statements that are true:
(a)	A bootstrap program is normally stored in ROM.
(b)	The main objective of batch multiprogramming systems is to minimize the CPU response time.
(c)	In uniprogramming, the processor can switch to another job while an I/O instruction is being executed.
(d)	In an Operating System, protected areas of memory can be accessed when the mode bit is 0.
(e)	Cache memory is typically faster, more expensive and has less storage space than the main memory.
(f)	A device driver moves the data between the I/O device that it controls and its local buffer storage.

Feedback:

A bootstrap program is normally stored in ROM.

The main objective of batch multiprogramming is to maximize processor usage. The response time is minimized in time sharing systems.

In uniprogramming, the processor must wait for the I/O instruction to conclude before proceeding.

In an Operating System, protected areas of memory can be accessed when the mode bit is 0 (kernel mode).

Cache memory is typically faster, more expensive and has less storage space than the main memory.

A device controller moves the data between the I/O device that it controls and its local buffer storage.

Q1.2	Select all the following statements that are true:
(a)	A bootstrap program is stored in RAM.
(b)	The main objective of batch multiprogramming systems is to maximize processor usage.
(c)	In uniprogramming, the processor must wait for the I/O instruction to conclude before proceeding.
(d)	In an Operating System, protected areas of memory can be accessed when the mode bit is 1.
(e)	Cache memory is typically faster, more expensive per byte and has less storage space than a magnetic disk.
(f)	A device controller provides the Operating System with a uniform interface to the IO device.

Feedback:

A bootstrap program is stored in a non-volatile memory, normally ROM or EEPROM.

The main objective of a batch multiprogramming system is to maximize processor usage.

In uniprogramming, the processor must wait for the I/O instruction to conclude before proceeding.

In an Operating System, protected areas of memory cannot be accessed in the user mode (when the mode bit is 1).

Cache memory is typically faster, more expensive per byte and has less storage space than a magnetic disk.

A *device driver* provides the Operating System with a uniform interface to the IO device.

Q1.3	Select all the following statements that are true:
(a)	A bootstrap program loads the OS kernel into memory.
(b)	The main objective of time-sharing systems is to maximize processor use.
(c)	In multiprogramming, the processor can switch to another job while an I/O instruction is being executed.
(d)	In an Operating System, certain areas of memory are protected when the mode bit set to 1.
(e)	RAM is typically cheaper, slower and has less storage space than a solid-state disk.
(f)	A device driver maintains a local buffer storage and a set of special-purpose registers.

Feedback:

A bootstrap program loads the OS kernel into memory.

The main objective of time-sharing systems is to minimize the CPU response time. The processor use is maximized in batch multi-programming systems.

In multiprogramming, the processor can switch to another job while an I/O instruction is being executed.

In an Operating System, certain areas of memory are protected when the mode bit set to 1 i.e. user mode.

RAM is typically more expensive, faster and has less storage space than a solid-state disk.

A *device controller* maintains some local buffer storage and a set of special-purpose registers.

Q2.1	Choose the correct statement(s):
(a)	A short-term scheduler selects which process should be executed next and allocates a CPU
(b)	A short-term scheduler must be considerably faster than a long-term scheduler.
(c)	A short-term scheduler is invoked infrequently as compared to a long-term scheduler.
(d)	A short-term scheduler controls the degree of multiprogramming in a system.
(e)	A long-term scheduler is invoked infrequently compared to a short-term scheduler.

Feedback:

Short-term Scheduler:

It is also known as **CPU scheduler**. Its main objective is to increase the system's performance in accordance with the chosen set of criteria (like Avg. RT / WT). CPU scheduler selects a process among the processes that are ready to execute and allocates a CPU to it for execution. Short-term

schedulers should be faster than long-term schedulers as they are invoked frequently as compared to long-term schedulers.

Long-term Scheduler:

It is also known as **Job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. The job scheduler controls the degree of multiprogramming and ensures a balanced mix of IO vs. CPU bound jobs in the system. The long-term scheduler is invoked infrequently as compared to the short-term scheduler; hence it can be much slower.

Q2.2	Choose the correct statement(s):
(a)	A CPU scheduler selects which process should be executed next and allocates a CPU.
(b)	A CPU scheduler must be considerably faster than a Job scheduler.
(c)	A CPU scheduler is invoked infrequently as compared to a Job scheduler.
(d)	A CPU scheduler controls the degree of multiprogramming in a system.
(e)	A Job scheduler is invoked infrequently compared to a CPU scheduler.

Feedback:

Short-term Scheduler:

It is also known as **CPU scheduler**. Its main objective is to increase the system's performance in accordance with the chosen set of criteria (like Avg. RT / WT). CPU scheduler selects a process among the processes that are ready to execute and allocates a CPU to it for execution. Short-term schedulers should be faster than long-term schedulers as they are invoked frequently as compared to long-term schedulers.

Long-term Scheduler:

It is also known as **Job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. The job scheduler controls the degree of multiprogramming and ensures a balanced mix of IO vs. CPU bound jobs in the system. The long-term scheduler is invoked infrequently as compared to the short-term scheduler; hence it can be much slower.

Q2.3	Choose the correct statement(s):
(a)	A long-term scheduler selects which process should be executed next and allocates a CPU.
(b)	A long-term scheduler selects which processes should be admitted into the ready queue.
(c)	A long-term scheduler is invoked infrequently compared to a short-term scheduler.
(d)	A long-term scheduler controls the degree of multiprogramming in a system.
(e)	A long-term scheduler is considerably faster as compared to a short-term scheduler.


Feedback:

Short-term Scheduler:

It is also known as **CPU scheduler**. Its main objective is to increase the system's performance in accordance with the chosen set of criteria (like Avg. RT / WT). CPU scheduler selects a process among the processes that are ready to execute and allocates a CPU to it for execution. Short-term schedulers should be faster than long-term schedulers as they are invoked frequently as compared to long-term schedulers.

Long-term Scheduler:

It is also known as **Job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. The job scheduler controls the degree of multiprogramming and ensures a balanced mix of IO vs. CPU bound jobs in the system. The long-term scheduler is invoked infrequently as compared to the short-term scheduler; hence it can be much slower.

Q3.1	<p>Consider a processor with 2 levels of memory, Level 1 Cache and Level 2 Main Memory (RAM), represented in the figure below. Level 1 has an access time of 5 ns and Level 2 has an access time of 0.2 μs. Assume that if a byte to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the byte is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the byte is in level 1 or level 2.</p> <p>We can define hit-ratio H, as the fraction of all memory accesses that are found in the faster memory (the cache memory). The hit-ratio H of cache is 97%. What is the average time to access a byte in such a two-level memory subsystem?</p> <div style="text-align: center;">  <pre> graph LR CPU[CPU] --- CM[Cache Memory] CM --- MM[Main Memory] </pre> </div>
(a)	10.85 ns
(b)	0.205 μ s
(c)	5.006 ns
(d)	11 ns
Feedback:	

The average access time is calculated as:

$$T_{Avg} = \text{Hit Rate} * \text{Cache access time} + \text{Miss Rate} * \text{Lower-level access time}$$

$$T_{Avg} = H \times T_1 + (1 - H) \times (T_1 + T_2) \quad \text{OR} \quad T_{Avg} = T_1 + (1 - H) \times T_2$$

We know that $H = 0.97$, $T_1 = 5 \text{ ns}$ and $T_2 = 0.2 \mu\text{s} = 200 \text{ ns}$

$$T_{Avg} = 0.97 \times 5 \text{ ns} + (1 - 0.97) \times (5 \text{ ns} + 200 \text{ ns})$$

$$= 4.85 \text{ ns} + 0.03 \times 205 \text{ ns} = 4.85 \text{ ns} + 6.15 \text{ ns} = 11 \text{ ns}$$

Q3.2 Consider a processor with 2 levels of memory, Level 1 Cache and Level 2 Main Memory (RAM), represented in the figure below. Level 1 has an access time of 11 ns and Level 2 has an access time of 0.18 μs . Assume that if a byte to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the byte is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the byte is in level 1 or level 2.

We can define hit-ratio H , as the fraction of all memory accesses that are found in the faster memory (the cache memory). The hit-ratio H of cache is 96%. What is the average time to access a byte in such a two-level memory subsystem?



(a) 17.76 ns

(b) 0.191 μs

(c) 11.007 ns

(d) 18.2 ns

Feedback:

The average access time is calculated as:


$$T_{Avg} = \text{Hit Rate} * \text{Cache access time} + \text{Miss Rate} * \text{Lower-level access time}$$

$$T_{Avg} = H \times T_1 + (1 - H) \times (T_1 + T_2) \quad \text{OR} \quad T_{Avg} = T_1 + (1 - H) \times T_2$$

We know that $H = 0.96$, $T_1 = 11 \text{ ns}$ and $T_2 = 0.18 \mu\text{s} = 180 \text{ ns}$

$$T_{Avg} = 0.96 \times 11 \text{ ns} + (1 - 0.96) \times (11 \text{ ns} + 180 \text{ ns})$$

$$= 10.56 \text{ ns} + 0.04 \times 191 \text{ ns} = 10.56 \text{ ns} + 7.64 \text{ ns} = 18.2 \text{ ns}$$

Q3.3	<p>Consider a processor with 2 levels of memory, Level 1 Cache and Level 2 Main Memory (RAM), represented in the figure below. Level 1 has an access time of 7 ns and Level 2 has an access time of 0.15 μs. Assume that if a byte to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the byte is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the byte is in level 1 or level 2. We can define hit-ratio H, as the fraction of all memory accesses that are found in the faster memory (the cache memory). The hit-ratio H of cache is 98%. What is the average time to access a byte in such a two-level memory subsystem?</p>  <pre> graph LR CPU[CPU] --- CM[Cache Memory] CM --- MM[Main Memory] </pre>
(a)	9.86 ns
(b)	0.157 μ s
(c)	7.003 ns
(d)	10 ns
<p>Feedback:</p> <p>The average access time is calculated as:</p> $T_{Avg} = \text{Hit Rate} * \text{Cache access time} + \text{Miss Rate} * \text{Lower-level access time}$ $T_{Avg} = H \times T_1 + (1 - H) \times (T_1 + T_2) \quad \text{OR} \quad T_{Avg} = T_1 + (1 - H) \times T_2$ <p>We know that $H = 0.98$, $T_1 = 7$ ns and $T_2 = 0.15 \mu\text{s} = 150$ ns</p> $T_{Avg} = 0.98 \times 7 \text{ ns} + (1 - 0.98) \times (7 \text{ ns} + 150 \text{ ns})$ $= 6.86 \text{ ns} + 0.02 \times 157 \text{ ns} = 6.86 \text{ ns} + 3.14 \text{ ns} = 10 \text{ ns}$	

Q4.1	<p>Consider the following program:</p> <pre> int main(){ printf("A"); fork(); printf("B"); fork(); printf("C"); fork(); fork(); return 0; } </pre> <p>How many times will the letters "A", "B" and "C" be printed?</p>
(a)	A:1 time, B: 2 times, C: 2 times
(b)	A:1 time, B: 2 times, C: 4 times
(c)	A:1 time, B: 1 time, C: 2 times

(d)	A:1 time, B: 2 times, C: 3 times
(e)	A:1 time, B: 1 time, C: 1 time
<p>Feedback:</p> <p>When a <code>fork()</code> system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from <code>fork()</code> call.</p> <p>The main idea is that the number of times “a letter” is printed is equal to the number of processes that will exist at that point where the letter is printed.</p> <ul style="list-style-type: none"> At the start there is only one process, so ‘A’ will be printed only one time because it is executed before making the first <code>fork()</code> system call. After the <code>fork()</code> there are two processes. ‘B’ will be printed 2 times. After the second <code>fork()</code> there will be 4 processes. ‘C’ will be printed 4 times – once in each process. There are two further <code>fork()</code> system calls but these won’t lead to any further output but will lead to the creation of further processes. <p>Therefore, the correct answer is A:1 time, B: 2 times, C: 4 times</p>	

Q4.2	<p>Consider the following program:</p> <pre>int main(){ printf("A"); fork(); printf("B"); fork(); fork(); printf("C"); fork(); return 0; }</pre> <p>How many times will the letters “A”, “B” and “C” be printed?</p>
(a)	A:1 time, B: 2 times, C: 2 times
(b)	A:1 time, B: 2 times, C: 4 times
(c)	A:1 time, B: 1 time, C: 2 times
(d)	A:1 time, B: 2 times, C: 8 times
(e)	A:4 times, B: 4 times, C: 4 times
<p>Feedback:</p> <p>When a <code>fork()</code> system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from <code>fork()</code> call.</p> <p>The main idea is that the number of times “a letter” is printed is equal to the number of processes that will exist at that point where the letter is printed.</p> <ul style="list-style-type: none"> At the start there is only one process, so ‘A’ will be printed only one time because it is executed 	

before making the first fork() system call.

- After the first fork() there are two processes. 'B' will be printed 2 times.
- After the second and third calls to fork() there will be 8 processes. 'C' will be printed 8 times – once in each process.
- There is a further fork() system call but this won't lead to any further output but will lead to the creation of further processes.

Therefore, the correct answer is A:1 time, B: 2 times, C: 8 times

Q4.3 Consider the following program:

```
int main(){
    printf("A");
    fork();
    fork();
    printf("B");
    fork();
    printf("C");
    fork();
    return 0;
}
```

How many times the letters "A", "B" and "C" will be printed?

- | | |
|-----|-----------------------------------|
| (a) | A:1 time, B: 2 times, C: 3 times |
| (b) | A:1 time, B: 4 times, C: 8 times |
| (c) | A:1 time, B: 2 times, C: 8 times |
| (d) | A:1 time, B: 4 times, C: 3 times |
| (e) | A:4 times, B: 4 times, C: 4 times |

Feedback:

When a fork() system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from fork() call.

The main idea is that the number of times "a letter" is printed is equal to the number of processes that will exist at that point where the letter is printed.

- At the start there is only one process, so 'A' will be printed only one time because it is executed before making the first fork() system call.
- After the two further calls to fork() there are 4 processes. 'B' will be printed 4 times.
- After the next call to fork() there will be 8 processes. 'C' will be printed 8 times – once in each process.
- There is a further fork() system call but this won't lead to any further output but will lead to the creation of further processes.

Therefore, the correct answer is A:1 time, B: 4 times, C: 8 times

Q5.1 Three processes (P1, P2 and P3) with the following arrival times and sequences of CPU and I/O bursts are presented below:

Process	Arrival time [ms]	CPU burst [ms]	I/O burst [ms]
P1	0	7	2
P2	0	3	3
P3	0	3	2

Assuming no time is wasted for context switching and the I/O burst starts immediately after a process finishes its CPU burst. Also, the I/O burst of a process can potentially overlap the CPU burst of another process. What will be the percentage of CPU utilization using the FCFS scheduling algorithm for this system?

(a) 100%

(b) 86.67%

(c) 65%

(d) 16%

Feedback:

P1	P2	P3	P3 I/O
0	7	10	13
	P1 I/O	---P2 I/O-----	

As illustrated in the Gantt chart, the total system utilization time = 15 ms
 CPU utilized for 13 ms (starting from 0 ms and ending at 13 ms).
 So, CPU utilization = $(13/15) \times 100 = 86.67\%$

Q5.2 Three processes (P1, P2 and P3) with the following arrival times and sequences of CPU and I/O bursts are presented below:

Process	Arrival time [ms]	CPU burst [ms]	I/O burst [ms]
P1	0	3	2
P2	0	5	1
P3	0	2	2

Assuming no time is wasted for context switching and the I/O burst starts immediately after a

	process finishes its CPU burst. Also, the I/O burst of a process can potentially overlap the CPU burst of another process. What will be the percentage of CPU utilization using the RR (with time quantum 5 ms) scheduling algorithm for this system?
(a)	100%
(b)	66.67%
(c)	16.67%
(d)	83.33%

Feedback:

P1	P2	P3	P3 I/O
0	3	8	10
	P1 I/O	P2 I/O	

As illustrated in the Gantt chart, the total system utilization time = 12 ms
CPU utilized for 10 ms (starting from 0 ms and ending at 10 ms).
So, CPU utilization = $(10/12) * 100 = 83.33\%$

Q5.3

Three processes (P1, P2 and P3) with the following arrival times and sequences of CPU and I/O bursts are presented below:

Process	Arrival time [ms]	CPU burst [ms]	I/O burst [ms]
P1	0	4	4
P2	0	3	1
P3	0	5	4

Assuming no time is wasted for context switching and the I/O burst starts immediately after a process finishes its CPU burst. Also, the I/O burst of a process can potentially overlap the CPU burst of another process. What will be the percentage of CPU utilization using non-preemptive SJF scheduling algorithm for this system?

(a)	75%
(b)	57.14%
(c)	100%
(d)	92.2%

Feedback:

P2	P1	P3	P3 I/O
0	3	7	12
	P2 I/O	--P1 I/O--	

As illustrated in the Gantt chart, the total system utilization time = 16 ms.
CPU utilized for 12 ms (starting from 0 ms and ending at 12 ms).
So, CPU utilization = $(12/16) * 100 = 75\%$

Q6.1 Consider the following algorithm:

```
for i=1..n {  
    for j=1..n {  
        for k=1..n/2 {  
            sum = sum + i + j + k  
        }  
    }  
}  
for i=1..n {  
    diff = diff - i  
}
```

Using Big-O notation, what is the time complexity of this algorithm?

(a) $O(n^3)$

(b) $O(1/2 n^3)$

(c) $O(1/2 n^3 + n)$

(d) $O(3n)$

(e) $O(n)$

Feedback:

The first part of the algorithm consists of 3 nested loops, and each of which is a function of n . It's complexity is, therefore, $O(n^3)$.

The second part is just one loop and has a complexity of $O(n)$.

When we combine these ($O(n^3) + O(n)$) we end up with $O(n^3)$ because as n gets large this part dominates.

Remember, we remove constants because they do not affect the way in which the amount of computation changes as n grows.

Q6.2	<p>Consider the following algorithm:</p> <pre> for i=1..n/2 { for j=1..n/3 { for k=1..n/4 { prod = prod + i * j * k } } } for i=1..n/2{ for j=1..n { sum = sum + i * j } } </pre> <p>Using Big-O notation, what is the time complexity of this algorithm?</p>
(a)	$O(n^3)$
(b)	$O(1/24 n^3)$
(c)	$O(1/24 n^3 + n^2)$
(d)	$O(1/24 n^3 + 1/2n^2)$
(e)	$O(n^2)$
<p>Feedback:</p> <p>The first part of the algorithm consists of 3 nested loops, and each of which is a function of n. It's complexity is, therefore, $O(n^3)$.</p> <p>The second part has 2 loops and has a complexity of $O(n^2)$.</p> <p>When we combine these ($O(n^3) + O(n^2)$) we end up with $O(n^3)$ because as n gets large this part dominates.</p> <p>Remember, we remove constants because they do not affect the way in which the amount of computation changes as n grows.</p>	

Q6.3	<p>Consider the following algorithm:</p> <pre> for i=1..n { for j=1..n { for k=1..10 { x = x + i + j + k } } } for i=1..2*n { for j=1..3*n { for k=1..20 { p = p + i * j * k } } } </pre> <p>Using Big-O notation, what is the time complexity of this algorithm?</p>
(a)	$O(n^2)$
(b)	$O(n^3)$
(c)	$O(n^3+6n^3)$
(d)	$O(6n^3)$
(e)	$O(2n^2)$
<p>Feedback:</p> <p>The first part of the algorithm consists of 3 nested loops. However, only the outer 2 loops are a function of n. It's complexity is, therefore, $O(n^2)$.</p> <p>The second part, again, has 3 nested loops with only the outer 2 loops are a function of n and it has a complexity of $O(n^2)$.</p> <p>When we combine these ($O(n^2) + O(n^2)$) we end up with $O(n^2)$ because it is the way it changes with the value of n that we are representing, not the actual time.</p> <p>Remember, we remove constants because they do not affect the way in which the amount of computation changes as n grows.</p>	

Q7.1	A program you have developed has a fixed startup time and a main algorithm which you know is $O(n^3)$. You have tested the program with 1 data item, and it takes 10 seconds to execute. With 100 data items it takes 15 seconds to run. Which of the following is the best estimate of how long (in seconds) it will take with 400 items?
(a)	330
(b)	320
(c)	60
(d)	30
(e)	960

Feedback:

With 1 item it takes 10 seconds. This means that it has a fixed startup cost of approximately 10 seconds (the time to process 1 item is negligible).

With $n=100$ it takes 15 seconds: 10 seconds startup time + 5 seconds to process the 100 items.

The main algorithm is $O(n^3)$.

Therefore, if we have 4 times as much data, we can estimate that it will take $4^3 = 64$ times as long to execute that part of the program + the fixed startup time.

[If we double the amount of data then we would increase the time 2^3 times = 8 times. If we doubled it again, it will be 4^3 times = 64 times and so on]

Therefore, it will take $10 + 64 \times 5 = 10 + 320 = 330$ seconds

Q7.2	A program you have developed has a fixed startup time and a main algorithm which you know is $O(n^3)$. You have tested the program with 1 data item, and it takes 20 seconds to execute. With 50 data items it takes 25 seconds to run. Which of the following is the best estimate of how long (in seconds) it will take with 200 items?
(a)	340
(b)	320
(c)	100
(d)	120
(e)	1600

Feedback:

With 1 item it takes 20 seconds. This means that it has a fixed startup cost of approximately 20 seconds (the time to process 1 item is negligible).

With $n=50$ it takes 25 seconds: 20 seconds startup time + 5 seconds to process the 50 items.

The main algorithm is $O(n^3)$.

Therefore, if we have 4 times as much data, we can estimate that it will take $4^3 = 64$ times as long to execute that part of the program + the fixed startup time.

[If we double the amount of data then we would increase the time 2^3 times = 8 times. If we doubled it again, it will be 4^3 times = 64 times and so on]

Therefore, it will take $20 + 64 \times 5 = 20 + 320 = 340$ seconds

Q7.3	A program you have developed has a fixed startup time and a main algorithm which you know is $O(n^3)$. You have tested the program with 1 data item, and it takes 5 seconds to execute. With 100 data items it takes 15 seconds to run. Which of the following is the best estimate of how long (in seconds) it will take with 800 items.
(a)	5125
(b)	5120
(c)	85
(d)	120
(e)	7680

Feedback:

With 1 item it takes 5 seconds. This means that it has a fixed startup cost of approximately 5 seconds (because the time to process 1 item is insignificant).

With $n=100$ it takes 15 seconds: 5 seconds startup time + 10 seconds to process the 100 items.

The main algorithm is $O(n^3)$.

Therefore, if we have 8 times as much data, we can estimate that it will take $8^3 = 512$ times as long to execute that part of the program + the fixed startup time.

[If we double the amount of data then we would increase the time 2^3 times = 8 times. If we doubled it again, it will be 4^3 times = 64 times and so on]

Therefore, it will take $5 + 512 \cdot 10 = 5 + 5120 = 5125$ seconds

Q8.1	<p>Consider the following class definition in Java (assuming all the necessary definitions are provided):</p> <pre> class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public static double getPerimeter(Rectangle rect) { double len = rect.getLength(); double wid = rect.getWidth(); double peri = 2 * (len + wid); return peri; } // some more code here } </pre>
------	---

	Which of the following show(s) the correct slot number allocation to the local variables in the <code>getPerimeter</code> method?
(a)	rect = 0, len = 1, wid = 3, peri = 5
(b)	rect = 0, len = 2, wid = 4, peri = 6
(c)	this = 0, rect = 1, len = 2, wid = 4, peri = 6
(d)	this = 0, rect = 1, len = 2, wid = 3, peri = 4
(e)	this = 0, rect = 1, len = 3, wid = 5, peri = 7
Feedback: The <code>getPerimeter</code> is a static method therefore, the 'this' reference will not exist. The parameter <code>rect</code> is of Reference type (4 bytes), therefore, it will take 1 slot. The local variables <code>len</code> , <code>wid</code> and <code>peri</code> are all double types (8 bytes), and each one will take 2 slots. Therefore, the correct answer is: rect = 0, len = 1, wid = 3, peri = 5	

Q8.2	Consider the following class definition in Java (assuming all the necessary definitions are provided): <pre> class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public static double getArea(Rectangle rect) { double len = rect.getLength(); double wid = rect.getWidth(); double area = len * wid; return area; } // some more code here } </pre> Which of the following show(s) the correct slot number allocation to the local variables in the <code>getArea</code> method?
(a)	rect = 0, len = 1, wid = 3, area = 5
(b)	rect = 0, len = 2, wid = 4, area = 6
(c)	this = 0, rect = 1, len = 2, wid = 4, area = 6
(d)	this = 0, rect = 1, len = 2, wid = 3, area = 4

(e)	this = 0, rect = 1, len = 3, wid = 5, area = 7
Feedback: getArea is a static method therefore, the 'this' reference will not exist. The parameter rect is of Reference type (4 bytes), therefore, it will take 1 slot. The local variables len, wid and area are all double types (8 bytes), and each one will take 2 slots. Therefore, the correct answer is: rect = 0, len = 1, wid = 3, area = 5	

Q8.3	<p>Consider the following class definition in Java (assuming all the necessary definitions are provided):</p> <pre> class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public Rectangle(double len, double wid, String col) { length = len; width = wid; color = col; count++; } // some more code here } </pre> <p>Which of the following show(s) the correct slot number allocation to the local variables in the Rectangle constructor?</p>
(a)	this = 0, len = 1, wid = 3, col = 5
(b)	this = 0, length = 1, width = 3, color = 5, count = 6
(c)	len = 0, wid = 2, col = 4
(d)	length = 0, width = 2, color = 4, count = 5
(e)	this = 0, len = 1, wid = 2, col = 3
Feedback: Rectangle is non-static, therefore the 'this' reference will exist and uses slot 0. The parameters len and wid are double types (8 bytes), and each one will take 2 slots. The parameter col is of reference type (4 bytes) so it will take 1 slot. Therefore, the correct answer is: this = 0, len = 1, wid = 3, col = 5	

Q9.1	<p>Consider the following class definition in Java (assuming all the necessary definitions are provided):</p> <pre> class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public double getPerimeter() { double len = getLength(); double wid = getWidth(); double peri = 2 * (len + wid); return peri; } // some more code here } </pre> <p>Identify the local, instance and class variables in the <code>getPerimeter</code> method and the class definition given above.</p>
(a)	<p>Local Variables: this, len, wid, & peri Instance Variables: length, width, color Class Variable: count</p>
(b)	<p>Local Variables: len, wid, & peri Instance Variables: length, width, color Class Variable: count</p>
(c)	<p>Local Variables: len, wid, & peri Instance Variables: this, length, width, color Class Variable: count</p>
(d)	<p>Local Variables: this, length, width, color Instance Variables: len, wid, & peri Class Variable: count</p>
<p>Feedback:</p> <p>The <code>getPerimeter</code> is a non-static method, therefore 'this' reference will exist.</p> <p>The <code>getPerimeter</code> method does not take any input parameters.</p> <p>The <code>getPerimeter</code> method declares three local variables i.e. len, wid, and peri.</p> <p>The class <code>Rectangle</code> declares three instance variables i.e. length, width and color, and one class variable count. So, the correct answer is:</p>	

Local Variables: this, len, wid, & peri
Instance Variables: length, width, color
Class Variable: count

Q9.2	<p>Consider the following class definition in Java (assuming all the necessary definitions are provided):</p> <pre> class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public static double getArea(Rectangle rect) { double len = rect.getLength(); double wid = rect.getWidth(); double area = len * wid; return area; } // some more code here } </pre> <p>Identify the local, instance and class variables in the getArea method and the class definition given above.</p>
(a)	<p>Local Variables: rect, len, wid, & area Instance Variables: length, width, color Class Variable: count</p>
(b)	<p>Local Variables: this, rect, len, wid, & area Instance Variables: length, width, color Class Variable: count</p>
(c)	<p>Local Variables: rect, len, wid, & area Instance Variables: this, length, width, color Class Variable: count</p>
(d)	<p>Local Variables: this, length, width, color Instance Variables: rect, len, wid, & area Class Variable: count</p>
<p>Feedback: The getArea is a static method, therefore 'this' reference will not exist.</p>	

The `getArea` method takes a `Rectangle` type parameter called 'rect'.

The `getArea` method declares three local variables i.e. `len`, `wid`, and `area`.

The class `Rectangle` declares three instance variables i.e. `length`, `width` and `color`, and one class variable `count`. So, the correct answer is:

Local Variables: `rect`, `len`, `wid`, & `area`

Instance Variables: `length`, `width`, `color`

Class Variable: `count`

Q9.3	<p>Consider the following class definition in Java (assuming all the necessary definitions are provided):</p> <pre>class Rectangle { private static int count = 0; private double length; private double width; private String color; // some more code here public Rectangle(double len, double wid, String col) { length = len; width = wid; color = col; count++; } // some more code here }</pre> <p>Identify the local, instance and class variables in the <code>Rectangle</code> constructor and the class definition given above.</p>
(a)	<p>Local Variables: <code>this</code>, <code>len</code>, <code>wid</code>, & <code>col</code></p> <p>Instance Variables: <code>length</code>, <code>width</code>, <code>color</code></p> <p>Class Variable: <code>count</code></p>
(b)	<p>Local Variables: <code>len</code>, <code>wid</code>, & <code>col</code></p> <p>Instance Variables: <code>length</code>, <code>width</code>, <code>color</code></p> <p>Class Variable: <code>count</code></p>
(c)	<p>Local Variables: <code>len</code>, <code>wid</code>, & <code>col</code></p> <p>Instance Variables: <code>this</code>, <code>length</code>, <code>width</code>, <code>color</code></p> <p>Class Variable: <code>count</code></p>
(d)	<p>Local Variables: <code>this</code>, <code>length</code>, <code>width</code>, <code>color</code></p>

	Instance Variables: len, wid, & col Class Variable: count
Feedback: The Rectang le constructor is non-static, therefore, 'this' reference will exist. The Rectang le constructor takes two parameters of double type i.e. 'len' and 'wid'. The Rectang le constructor takes another parameter of String type i.e. 'col'. The Rectang le constructor does not declare any other local variables. The class Rectang le declares three instance variables i.e. length, width and color, and one class variable count. So, the correct answer is: Local Variables: this, len, wid, & col Instance Variables: length, width, color Class Variable: count	

Q10.1

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed and is assigned a priority between 1-10 (lower numbers mean higher priority). In answering the question, base all decisions on the information you have at the time the decision must be made. Do not count the context switches at time zero and at the end.

Process	Arrival time [ms]	Burst time [ms]	Priority
P1	0.0	8	10
P2	0.4	4	2
P3	0.5	1	10
P4	0.8	2	1
P5	1.0	2	5

Which of these statement(s) illustrate(s) the execution of these processes using preemptive priority (SJF if priority is equal)?

(a)

At 9 ms, process P3 is in the running state.

(b)

The number of context switches needed here is 6.

(c)

The average waiting time across all processes = 8.26 ms.

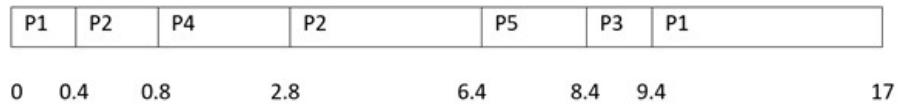
(d)

The average turnaround time across all processes = 4.86 ms.

(e)

Process P5 finishes its execution after all other processes.

Feedback:



Process	Arrival time [ms]	Burst time [ms]	Turnaround time (waiting time+ service time) [ms]	Waiting time [ms]
P1	0.0	8	17	9
P2	0.4	4	6	2
P3	0.5	1	8.9	7.9
P4	0.8	2	2	0
P5	1.0	2	7.4	5.4

The average waiting time across all processes = $24.3/5 = 4.86$ ms

The average turnaround time across all processes = $41.3/5 = 8.26$ ms

As shown in the Gantt chart, process P3 is in the running state at 9 ms

Process P1 finishes its execution after all other processes and the number of context switches needed here is 6.

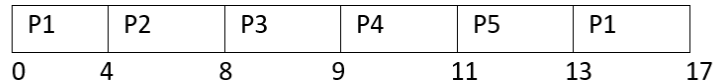
Q10.2 Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed and is assigned a priority between 1-10 (lower numbers mean higher priority). In answering the question, base all decisions on the information you have at the time the decision must be made. Do not count the context switches at time zero and at the end.

Process	Arrival time [ms]	Burst time [ms]	Priority
P1	0.0	8	10
P2	0.4	4	2
P3	0.5	1	10
P4	0.8	2	1
P5	1.0	2	5

Which of these statement(s) illustrate(s) the execution of these processes using RR (with time quantum 4ms)?

(a)	At 12 ms, process P5 is in the running state.
(b)	The number of context switches needed here is 4.
(c)	The average waiting time across all processes = 7.66 ms.
(d)	The average turnaround time across all processes = 9 ms.
(e)	Process P2 finishes its execution after all other processes.

Feedback:



Process	Arrival time [ms]	Burst time [ms]	Turnaround time (waiting time+ service time) [ms]	Waiting time [ms]
P1	0.0	8	17	9
P2	0.4	4	7.6	3.6
P3	0.5	1	8.5	7.5
P4	0.8	2	10.2	8.2
P5	1.0	2	12	10

The average waiting time across all processes = $38.3 / 5 = 7.66$ ms

The average turnaround time across all processes = $55.3 / 5 = 11.06$ ms

As shown in the Gantt chart, process P5 is in the running state at 12 ms

Process P1 finishes its execution after all other processes and the number of context switches needed here is 5.

Q10.3

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed and is assigned a priority between 1-10 (lower numbers mean higher priority). In answering the question, base all decisions on the information you have at the time the decision must be made. Do not count the context switches at time zero and at the end.

Process	Arrival time [ms]	Burst time [ms]	Priority
P1	0.0	8	10
P2	0.4	4	2
P3	0.5	1	10
P4	0.8	2	1

	P5	1.0	2	5
--	----	-----	---	---

Which of these statement(s) illustrate(s) the execution of these processes using FCFS?

- (a) At 14 ms, process P4 is in the waiting (ready) state.
- (b) The number of context switches needed here is 5.
- (c) The average waiting time across all processes = 7.06 ms.
- (d) The average turnaround time across all processes = 12.46 ms.
- (e) Process P5 finishes its execution after all other processes.

Feedback:

P1	P2	P3	P4	P5	
0	8	12	13	15	17

Process	Arrival time [ms]	Burst time [ms]	Turnaround time (waiting time+ service time) [ms]	Waiting time [ms]
P1	0.0	8	8	0
P2	0.4	4	11.6	7.6
P3	0.5	1	12.5	11.5
P4	0.8	2	14.2	12.2
P5	1.0	2	16	14

The average waiting time across all processes = $45.3/5 = 9.06$ ms

The average turnaround time across all processes = $62.3/5 = 12.46$ ms

As shown in the Gantt chart, process P4 is in the running state (not waiting state) at 14 ms

Process P5 finishes its execution after all other processes and the number of context switches needed here is 4.