Study Guide Questions with Answers - Part 2 of 2

Searching and Sorting in C# + Algorithm Efficiency

1. In the context of implementing `IComparable` for a custom class, what is the purpose of the `CompareTo` method?

- a) To determine if two objects are equal based on their properties.
- b) To compare two objects and return a value indicating their relative order.
- c) To perform a deep comparison of two objects, including their nested properties.
- d) To provide a mechanism for sorting objects using the `Sort` method of the `List` class.

Answer: b

Analysis:

- **Option a)** is incorrect because `CompareTo` is for determining relative order, not equality.
- **Option b)** is correct because `CompareTo` returns a value indicating whether the instance precedes, follows, or occurs in the same position in the sort order as the specified object.
- **Option c)** is incorrect because `CompareTo` does not perform deep comparisons.
- **Option d)** is partially correct but does not capture the primary purpose as accurately as b).

2. In the context of the `Array` class in C#, what does the `Rank` property represent?

- a) The number of elements in the array.
- b) The dimension of the array.
- c) The maximum index of the array.
- d) The minimum index of the array.

Answer: b

Analysis:

- **Option a)** is incorrect because `Rank` represents the number of dimensions.
- **Option b)** is correct because `Rank` represents the number of dimensions of the array.
- **Option c)** is incorrect because it describes the maximum index, not the number of dimensions.
- **Option d)** is incorrect because it describes the minimum index, not the number of dimensions.

3. Which sorting algorithm is often used in `Array.Sort()` for large arrays?

- a) Bubble Sort
- b) Insertion Sort
- c) Merge Sort

- d) Quick Sort

Answer: d

Analysis:

- **Option a)** is incorrect because Bubble Sort is inefficient for large arrays.
- **Option b)** is incorrect because Insertion Sort is also inefficient for large arrays.
- **Option c)** is partially correct but not the most commonly used in `Array.Sort()`.
- **Option d)** is correct because Quick Sort is often used due to its good average-case performance.

4. Given the following code snippet, which sorting algorithm is being implemented?

```
public void Sort(int[] array)
{
    for (int i = 0; i < array.Length - 1; i++)
    {
        for (int j = 0; j < array.Length - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                (array[j], array[j + 1]) = (array[j + 1], array[j]);
            }
        }
    }
}
```

- a) Bubble Sort
- b) Insertion Sort
- c) Merge Sort
- d) Selection Sort

Answer: a

Analysis:

- **Option a)** is correct because the algorithm described is Bubble Sort.
- **Option b)** is incorrect because the algorithm does not match Insertion Sort.
- **Option c)** is incorrect because the algorithm does not match Merge Sort.
- **Option d)** is incorrect because the algorithm does not match Selection Sort.

5. Which of the following is NOT a feature of the `Array` class in C#?

- a) Searching
- b) Sorting
- c) Reversing elements

- d) Dynamic resizing

Answer: d

Analysis:

- **Option a)** is a feature of the `Array` class.
- **Option b)** is a feature of the `Array` class.
- **Option c)** is a feature of the `Array` class.
- **Option d)** is correct because arrays in C# have a fixed size once created and cannot be dynamically resized.

6. Why is MergeSort still used today?

- a) It guarantees the smallest possible memory usage.
- b) It boasts exceptional performance on nearly sorted inputs.
- c) It offers superior performance in distributed computing environments.
- d) It simplifies implementation in resource-constrained environments.

Answer: c

Analysis:

- **Option a)** is incorrect because Merge Sort does not guarantee the smallest possible memory usage.
- **Option b)** is incorrect because Merge Sort's performance on nearly sorted inputs is not its primary advantage.
- **Option c)** is correct because Merge Sort offers superior performance in parallel and distributed computing environments due to its ability to be easily parallelized.
- **Option d)** is incorrect because Merge Sort is not simpler to implement in resource-constrained environments compared to other algorithms.

7. Which of the following best describes how Heap Sort works?

- a) It iteratively selects elements using a specialized data structure and places them in the sorted portion.
- b) It employs a divide-and-conquer strategy, recursively breaking down the array into smaller segments and sorting them individually.
- c) It iteratively selects the smallest element from the unsorted portion of the array and swaps it with the element at the beginning of the unsorted portion.
- d) It rearranges elements based on their positions in the array, starting from the middle and working outwards.

Answer: a

Analysis:

- **Option a)** is correct because Heap Sort uses a heap data structure to iteratively select and sort elements.
- **Option b)** describes Merge Sort or Quick Sort, not Heap Sort.
- **Option c)** describes Selection Sort, not Heap Sort.
- **Option d)** does not accurately describe any standard sorting algorithm.

8. What is a major advantage of Quick Sort over Merge Sort?

- a) Quick Sort has a better worst-case time complexity.
- b) Quick Sort is more stable than Merge Sort.
- c) Quick Sort typically has better average-case performance.
- d) Quick Sort can easily be parallelized.

Answer: c

Analysis:

- **Option a)** is incorrect because Merge Sort has a better worst-case time complexity (O(n log n)).
- **Option b)** is incorrect because Quick Sort is not stable.
- **Option c)** is correct because Quick Sort typically performs better on average compared to Merge Sort.
- **Option d)** is incorrect because Merge Sort is more easily parallelized.

9. What is the main difference between Depth-First Search (DFS) and Breadth-First Search (BFS) in tree traversal?

- a) DFS uses a queue, while BFS uses a stack.
- b) DFS uses a stack, while BFS uses a queue.
- c) DFS is used for sorting, while BFS is used for searching.
- d) DFS is faster than BFS in all cases.

Answer: b

Analysis:

- **Option a)** is incorrect because it reverses the data structures used by DFS and BFS.
- **Option b)** is correct because DFS uses a stack and BFS uses a queue.
- **Option c)** is incorrect because both DFS and BFS are used for searching.
- **Option d)** is incorrect because the performance depends on the specific use case.

10. What is the primary advantage of using Depth-First Search (DFS) over Breadth-First Search (BFS)?

- a) DFS guarantees finding the shortest path.
- b) DFS is always faster than BFS.
- c) DFS can handle weighted graphs.
- d) DFS uses less memory in most cases.

Answer: d

Analysis:

- **Option a)** is incorrect because BFS guarantees finding the shortest path in unweighted graphs.
- **Option b)** is incorrect because DFS is not always faster; it depends on the specific problem.
- **Option c)** is incorrect because both DFS and BFS can handle weighted graphs with modifications.
- **Option d)** is correct because DFS typically uses less memory compared to BFS.

11. Which of the following LINQ methods performs a filtering operation on a collection?

- a) Select
- b) Where
- c) OrderBy
- d) GroupBy

Answer: b

Analysis:

- **Option a)** performs a projection operation.
- **Option b)** is correct because it performs a filtering operation.
- **Option c)** performs a sorting operation.
- **Option d)** performs a grouping operation.

12. What is the result of the following LINQ query?

```
List<int> numbers = new() { 5, 3, 9, 1, 6 };
var result = numbers.OrderBy(n => n).First();
```

- a) 5
- b) 3
- c) 9
- d) 1

Answer: d

Analysis:

- **Option a)** is incorrect because `5` is not the first element after sorting the list in ascending order.
- **Option b)** is incorrect because `3` is not the first element after sorting the list in ascending order.
- **Option c)** is incorrect because `9` is the last element, not the first element after sorting the list in ascending order.
- **Option d)** is correct because after sorting the list `{5, 3, 9, 1, 6}` in ascending order, it becomes `{1, 3, 5, 6, 9}`, and the `First` method returns the first element of the sorted list, which is `1`.

13. Given the following code snippet, what will be printed after executing this code?

```
Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();
// Simulate code section 1 execution time
Thread.Sleep(150);
stopwatch.Stop();
Console.WriteLine($"Section 1 time: {stopwatch.ElapsedMilliseconds} ms");

stopwatch.Restart();
// Simulate code section 2 execution time
Thread.Sleep(250);
stopwatch.Stop();
Console.WriteLine($"Section 2 time: {stopwatch.ElapsedMilliseconds} ms");
```

- a) Section 1 time: 150 ms, Section 2 time: 250 ms
- b) Section 1 time: 150 ms, Section 2 time: 400 ms
- c) Section 1 time: 250 ms, Section 2 time: 150 ms
- d) Section 1 time: 150 ms, Section 2 time: 450 ms

Answer: a

Analysis:

- **Option a)** is correct because `Stopwatch.Restart` resets the elapsed time, so Section 1 is approximately 150 ms and Section 2 is approximately 250 ms.
- **Option b)** is incorrect because the times are not cumulative.
- **Option c)** is incorrect because the times are swapped.
- **Option d)** is incorrect because the times are not cumulative.

Streams

1. Which of the following classes provides methods for reading and writing to files using byte streams in C#?

- a) File

- b) Stream
- c) FileStream
- d) MemoryStream

Answer: c

Analysis:

- **Option a)** is incorrect because `File` provides static methods for file operations.
- **Option b)** is incorrect because `Stream` is an abstract base class.
- **Option c)** is correct because `FileStream` is used for reading and writing to files using byte streams.
- **Option d)** is incorrect because `MemoryStream` operates on memory, not files.

2. Which class in C# is typically used for reading text files line by line?

- a) FileStream
- b) StreamReader
- c) BinaryReader
- d) MemoryStream

Answer: b

Analysis:

- **Option a)** is incorrect because `FileStream` works with byte streams.
- **Option b)** is correct because `StreamReader` is typically used for reading text files line by line.
- **Option c)** is incorrect because `BinaryReader` reads binary data.
- **Option d)** is incorrect because `MemoryStream` operates on memory, not files.

3. Given the following code snippet, what will be the content of the file after execution?

```
using (FileStream fs = new FileStream("example.bin", FileMode.Create))
using (BinaryWriter writer = new BinaryWriter(fs))
{
    writer.Write(1234);
    writer.Write("Hello");
    writer.Write(true);
}
```

- a) The file will contain the string "1234HelloTrue".
- b) The file will contain data representing the integer 1234, the string "Hello", and the boolean true.
- c) The file will be empty because it only writes to memory.

- d) The file will contain only the string "Hello" because it overrides previous content with each write operation.

Answer: b

Analysis:

- **Option a)** is incorrect because the file contains binary data, not a concatenated string.
- **Option b)** is correct because the file contains the binary representation of an integer, a string, and a boolean.
- **Option c)** is incorrect because the file is not empty.
- **Option d)** is incorrect because all data is written sequentially without overwriting.

4. Which method is used to move the position within a stream to a specified location in C#?

- a) MoveTo
- b) SetPosition
- d) Position
- c) Seek

Answer: d

Analysis:

- **Option a)** is incorrect because there is no `MoveTo` method.
- **Option b)** is incorrect because there is no `SetPosition` method.
- **Option c)** is incorrect because `Position` is a property, not a method.
- **Option d)** is correct because `Seek` moves the position within a stream.

5. Which of the following methods is used to read binary data from a stream in C#?

- a) Read
- b) ReadBytes
- c) ReadBinary
- d) ReadBuffer

Answer: b

Analysis:

- **Option a)** is incorrect because `Read` reads byte arrays, not specifically binary data.
- **Option b)** is correct because `ReadBytes` reads a specified number of bytes from the stream.
- **Option c)** is incorrect because there is no `ReadBinary` method.
- **Option d)** is incorrect because there is no `ReadBuffer` method.

6. What is the purpose of the `Flush` method in stream handling in C#?

- a) To clear the buffer and write any buffered data to the underlying device.
- b) To reset the position of the stream to the beginning.
- c) To close the stream.
- d) To convert the stream to a different format.

Answer: a

Analysis:

- **Option a)** is correct because `Flush` clears the buffer and writes buffered data to the underlying device.
- **Option b)** is incorrect because `Flush` does not reset the stream position.
- **Option c)** is incorrect because `Flush` does not close the stream.
- **Option d)** is incorrect because `Flush` does not convert the stream.

7. Which of the following correctly describes how to catch filesystem events in C#?

- a) Using the `FileSystemMonitor` class to monitor file and directory changes.
- b) Using the `DirectoryWatcher` class to catch file and directory modifications.
- c) Using the `FileMonitor` class to track file and directory modifications.
- d) Using the `FileSystemWatcher` class to monitor files and directory changes.

Answer: d

Analysis:

- **Option a)** is incorrect because there is no `FileSystemMonitor` class.
- **Option b)** is incorrect because there is no `DirectoryWatcher` class.
- **Option c)** is incorrect because there is no `FileMonitor` class.
- **Option d)** is correct because `FileSystemWatcher` is used to listen for file and directory changes, allowing for event handling on these changes.

8. Given the following code snippet, what does the `Seek` method do?

```
using (FileStream fs = new FileStream("example.bin", FileMode.Open))
{
    fs.Seek(10, SeekOrigin.Begin);
    int data = fs.ReadByte();
}
```

- a) It sets the position within the stream to the 10th byte from the beginning, allowing reading from that position onward.

- b) It sets the position within the stream to the 10th byte from the end, reading backwards.
- c) It moves the current position within the stream 10 bytes ahead from the current position, reading the next byte.
- d) It moves the current position within the stream 10 bytes backward from the current position, reading the previous byte.

Answer: a

Analysis:

- **Option a)** is correct because `Seek(10, SeekOrigin.Begin)` sets the position to the 10th byte from the beginning.
- **Option b)** is incorrect because it does not set the position from the end.
- **Option c)** is incorrect because it does not move 10 bytes ahead from the current position.
- **Option d)** is incorrect because it does not move 10 bytes backward from the current position.

9. What is the difference between named and anonymous pipes in C#?

- a) Named pipes can only be used for inter-thread communication, while anonymous pipes can be used for inter-process communication.
- b) Named pipes are identified by a specific name and can be used for inter-process communication, while anonymous pipes do not have a name and are typically used for inter-thread communication.
- c) Named pipes are faster than anonymous pipes.
- d) Anonymous pipes provide more security than named pipes.

Answer: b

Analysis:

- **Option a)** is incorrect because it reverses the usage of named and anonymous pipes.
- **Option b)** is correct because named pipes are identified by a name and can be used for inter-process communication, while anonymous pipes are used for inter-thread communication.
- **Option c)** is incorrect because performance is not a distinguishing factor.
- **Option d)** is incorrect because security features are not a primary distinction.

10. How does the `PipeStream` class in C# facilitate inter-process communication?

- a) By allowing the reading and writing of data to and from memory buffers.
- b) By enabling data transfer between the memory and a file.
- c) By providing a means for different processes to read and write data to the same named or unnamed pipe, facilitating communication.
- d) By converting text data to binary data for communication between processes.

Answer: c

Analysis:

- **Option a)** is incorrect because `PipeStream` is not specifically for memory buffers.
- **Option b)** is incorrect because it does not involve file transfer.
- **Option c)** is correct because `PipeStream` allows inter-process communication through named or unnamed pipes.
- **Option d)** is incorrect because data conversion is not the primary purpose.

11. In the context of stream processing, what is the primary advantage of using `BufferedStream` in C#?

- a) To handle network communications.
- b) To provide a buffer for read and write operations, reducing the number of I/O operations and improving performance.
- c) To read and write text data efficiently.
- d) To encrypt data before writing it to the underlying stream.

Answer: b

Analysis:

- **Option a)** is incorrect because `BufferedStream` is not specifically for network communications.
- **Option b)** is correct because `BufferedStream` improves performance by reducing the number of I/O operations.
- **Option c)** is incorrect because `BufferedStream` is not specifically for text data.
- **Option d)** is incorrect because `BufferedStream` does not handle encryption.

12. Given the following code snippet, what does the `Flush` method do?

```csharp
using (FileStream fs = new FileStream("example.bin", FileMode.Create))
using (BufferedStream bs = new BufferedStream(fs))
using (StreamWriter writer = new StreamWriter(bs))
{
    writer.WriteLine("Hello, World!");
    writer.Flush();
}
```

- a) It closes the stream.
- b) It clears the buffer and discards the buffered data.
- c) It resets the stream's position to the beginning.
- d) It writes any buffered data to the underlying device.

Answer: d

Analysis:

- **Option a)** is incorrect because `Flush` does not reset the position.
- **Option b)** is correct because `Flush` writes buffered data to the underlying device.
- **Option c)** is incorrect because `Flush` does not close the stream.
- **Option d)** is incorrect because `Flush` does not discard data.

13. What is a key characteristic of named pipes in C#?

- a) They are identified by a unique name and can be used for communication between different processes, even on different machines.
- b) They do not have a name and are typically used for communication between threads within the same process.
- c) They are slower than anonymous pipes.
- d) They provide more security features than anonymous pipes.

Answer: a

Analysis:

- **Option a)** is correct because named pipes can be used for inter-process communication across different machines.
- **Option b)** is incorrect because it describes anonymous pipes.
- **Option c)** is incorrect because performance is not a distinguishing factor.
- **Option d)** is incorrect because security is not the primary characteristic.

14. How do you set a timeout for read and write operations on a stream in C#?

- a) Using the `Timeout` property.
- b) Using the `SetTimeout` method.
- c) Using the `ReadTimeout` and `WriteTimeout` properties.
- d) Using the `ConfigureTimeout` method.

Answer: c

Analysis:

- **Option a)** is incorrect because there is no `Timeout` property.
- **Option b)** is incorrect because there is no `SetTimeout` method.
- **Option c)** is correct because `ReadTimeout` and `WriteTimeout` properties set timeouts.
- **Option d)** is incorrect because there is no `ConfigureTimeout` method.

15. Given the following code snippet, what will be the output of the `Console.WriteLine` statement?

```csharp
byte[] data = { 1, 2, 3, 4, 5 };
using (MemoryStream ms = new MemoryStream(data))
using (BinaryWriter writer = new BinaryWriter(ms))
using (BinaryReader reader = new BinaryReader(ms))
{
    writer.Seek(2, SeekOrigin.Begin);
    writer.Write((byte)9);
    ms.Position = 0;
    string output = "";
    for (int i = 0; i < data.Length; i++)
    {
        output += reader.ReadByte() + " ";
    }
    Console.WriteLine(output.Trim());
}
```

- a) "1 2 3 4 5"
- b) "1 2 9 4 5"
- c) "1 9 3 4 5"
- d) "1 2 3 9 5"

Answer: b

Analysis:

- **Option a)** is incorrect because the data is modified.
- **Option b)** is correct because the third byte is replaced with 9.
- **Option c)** is incorrect because it does not match the modification.
- **Option d)** is incorrect because it does not match the modification.

16. Given the following code snippet, what will happen if "destination.txt" already exists?

```csharp
File.Copy("source.txt", "destination.txt", true);
```

- a) "source.txt" will be moved to "destination.txt".
- b) "destination.txt" will be overwritten by "source.txt".
- c) An exception will be thrown.
- d) The contents of "source.txt" will be appended to "destination.txt".

Answer: b

Analysis:

- **Option a)** is incorrect because `File.Copy` does not move files.
- **Option b)** is correct because `true` allows overwriting.

- **Option c)** is incorrect because no exception is thrown if overwriting is allowed.
- **Option d)** is incorrect because `File.Copy` does not append content.

17. What is the purpose of the `BinaryReader` class in C#?

- a) To read primitive data types as binary values from a stream.
- b) To read text data from a stream.
- c) To read and write data to a file.
- d) To monitor changes in the filesystem.

Answer: a

Analysis:

- **Option a)** is correct because `BinaryReader` reads primitive data types as binary values.
- **Option b)** is incorrect because `BinaryReader` is not for text data.
- **Option c)** is incorrect because it does not write data.
- **Option d)** is incorrect because it does not monitor the filesystem.

18. Which method would you use to delete a directory that is not empty in C#?

- a) Directory.Delete(path);
- b) Directory.Remove(path);
- c) DirectoryInfo.Delete(true);
- d) DirectoryInfo.Remove(true);

Answer: c

Analysis:

- **Option a)** is incorrect because `Directory.Delete` does not handle non-empty directories without recursion.
- **Option b)** is incorrect because there is no `Directory.Remove` method.
- **Option c)** is correct because `DirectoryInfo.Delete(true)` deletes non-empty directories.
- **Option d)** is incorrect because there is no `DirectoryInfo.Remove` method.

19. How does the `Directory` class differ from the `DirectoryInfo` class in C#?

- a) `Directory` provides static methods for directory operations, while `DirectoryInfo` provides instance methods.
- b) `Directory` is used for file operations, while `DirectoryInfo` is used for network operations.
- c) `Directory` can only read directories, while `DirectoryInfo` can also write to directories.
- d) `Directory` handles binary data, while `DirectoryInfo` handles text data.

Answer: a

Analysis:

- **Option a)** is correct because `Directory` provides static methods and `DirectoryInfo` provides instance methods.
- **Option b)** is incorrect because both are used for directory operations.
- **Option c)** is incorrect because both can read and write directories.
- **Option d)** is incorrect because neither specifically handles binary or text data.

20. Given the following code snippet, what will be the output of the `Console.WriteLine` statement?

```
byte[] byteArray = { 1, 2, 3, 4, 5 };
using (MemoryStream ms = new MemoryStream(byteArray))
using (BinaryReader reader = new BinaryReader(ms))
{
    ms.Seek(2, SeekOrigin.Begin);
    byte b = reader.ReadByte();
    Console.WriteLine(b);
}
```

- a) 1
- b) 2
- c) 3
- d) 4

Answer: c

Analysis:

- **Option a)** is incorrect because the position is moved to the third byte.
- **Option b)** is incorrect because the position is moved to the third byte.
- **Option c)** is correct because `Seek(2, SeekOrigin.Begin)` moves to the third byte.
- **Option d)** is incorrect because the third byte is read.

Multithreading and Asynchronous Programming

1. What is the primary benefit of using the `ThreadPool` class in C#?

- a) Manual management of threads.
- b) Ensures threads execute in FIFO order.
- c) Guarantees single-thread execution.
- d) Reusable threads for multiple tasks.

Answer: d

Analysis:

- **Option a)** is incorrect because the `ThreadPool` handles thread management automatically.
- **Option b)** is incorrect because `ThreadPool` does not guarantee FIFO order.
- **Option c)** is incorrect because `ThreadPool` does not guarantee single-thread execution.
- **Option d)** is correct because the `ThreadPool` reuses threads for multiple tasks, which improves performance by reducing the overhead of creating and destroying threads.

2. Given the following code snippet, what is the likely outcome of using `thread.Join()` in this context?

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Thread thread = new Thread(PrintNumbers);
        thread.Start();
        thread.Join();
        Console.WriteLine("Thread has finished executing.");
    }

    static void PrintNumbers()
    {
        for (int i = 1; i <= 3; i++)
        {
            Console.WriteLine(i);
            Thread.Sleep(100); // Simulate work
        }
    }
}
```

- a) The program will throw an exception.
- b) The main thread continues running without waiting.
- c) The main thread waits until `PrintNumbers` finishes, then prints the message.
- d) The `Join` method pauses `PrintNumbers` until the main thread finishes.

Answer: c

Analysis:

- **Option a)** is incorrect because `thread.Join()` will not throw an exception in this context.
- **Option b)** is incorrect because `thread.Join()` causes the main thread to wait for `PrintNumbers` to finish.

- **Option c)** is correct because `thread.Join()` makes the main thread wait until `PrintNumbers` completes, and then prints the message.
- **Option d)** is incorrect because `thread.Join()` does not pause the `PrintNumbers` method.

3. How does the `Monitor.Enter` and `Monitor.Exit` mechanism enhance thread safety in C#?

- a) Ensures only one thread can access an object at a time.
- b) Allows a thread to enter a sleeping state indefinitely until manually interrupted.
- c) Starts and stops a thread, managing the lifecycle automatically.
- d) Schedules threads in the thread pool to optimize CPU usage.

Answer: a

Analysis:

- **Option a)** is correct because `Monitor.Enter` and `Monitor.Exit` ensure that only one thread can access a critical section of code at a time.
- **Option b)** is incorrect because it describes the behavior of `Thread.Sleep`.
- **Option c)** is incorrect because `Monitor` does not manage thread lifecycle.
- **Option d)** is incorrect because `Monitor` does not schedule threads.

4. Given the following code snippet, what will be the result?

```csharp
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        await RunTasks();
        Console.WriteLine("Tasks completed.");
    }

    static async Task RunTasks()
    {
        var task1 = Task.Run(() => DoWork("Task 1"));
        var task2 = Task.Run(() => DoWork("Task 2"));

        await Task.WhenAll(task1, task2);
    }

    static void DoWork(string taskName)
    {
        Console.WriteLine($"{taskName} is running.");
        Task.Delay(500).Wait();
    }
}
```

- a) "Tasks completed." is printed before "Task 1 is running." and "Task 2 is running."
- b) "Tasks completed." is printed after both "Task 1 is running." and "Task 2 is running."
- c) "Tasks completed." is printed after "Task 1 is running." but before "Task 2 is running."
- d) An exception is thrown because `Task.WhenAll` is not awaited properly.

Answer: b

Analysis:

- **Option a)** is incorrect because `Task.WhenAll` ensures "Tasks completed." is printed after both tasks are complete.
- **Option b)** is correct because `Task.WhenAll` waits for both tasks to complete before printing "Tasks completed."
- **Option c)** is incorrect because `Task.WhenAll` ensures both tasks complete first.
- **Option d)** is incorrect because `Task.WhenAll` is awaited properly.

5. How do `async` and `await` improve the responsiveness of a UI application?

- a) By converting asynchronous code into synchronous code.
- b) By blocking the main thread until all tasks are complete.
- c) By not blocking the main thread.
- d) By running all tasks on separate threads.

Answer: c

Analysis:

- **Option a)** is incorrect because `async` and `await` are used to maintain asynchronous code.
- **Option b)** is incorrect because they do not block the main thread.
- **Option c)** is correct because `async` and `await` allow the main thread to remain responsive while awaiting tasks.
- **Option d)** is incorrect because `async` and `await` do not necessarily run tasks on separate threads.

6. How does the `Task.Delay` method differ from `Thread.Sleep` in terms of asynchronous programming?

- a) `Task.Delay` blocks the current thread while `Thread.Sleep` does not.
- b) `Thread.Sleep` blocks the current thread while `Task.Delay` does not.
- c) `Task.Delay` is for CPU-bound operations, while `Thread.Sleep` is for I/O-bound operations.
- d) `Task.Delay` stops all tasks, while `Thread.Sleep` stops the main thread only.

Answer: b

Analysis:

- **Option a)** is incorrect because `Task.Delay` does not block the current thread.
- **Option b)** is correct because `Thread.Sleep` blocks the current thread, whereas `Task.Delay` does not.
- **Option c)** is incorrect because the purpose of `Task.Delay` and `Thread.Sleep` is not related to CPU-bound or I/O-bound operations.
- **Option d)** is incorrect because neither `Task.Delay` nor `Thread.Sleep` stops all tasks or only the main thread.

7. What does the `AggregateException` class in C# do?

- a) Handles file system exceptions.
- b) Handles multiple exceptions into a single exception.
- c) Manages thread synchronization.
- d) Combines data from multiple sources.

Answer: b

Analysis:

- **Option a)** is incorrect because `AggregateException` is not specific to file system exceptions.
- **Option b)** is correct because `AggregateException` represents one or more errors that occur during application execution.
- **Option c)** is incorrect because `AggregateException` does not manage thread synchronization.
- **Option d)** is incorrect because `AggregateException` does not combine data.

8. What is the difference between `Task.WhenAll` and `Task.WaitAll` in C#?

- a) `Task.WhenAll` waits for all tasks to complete asynchronously, while `Task.WaitAll` blocks the calling thread until all tasks complete.
- b) `Task.WaitAll` waits for all tasks to complete asynchronously, while `Task.WhenAll` blocks the calling thread until all tasks complete.
- c) `Task.WhenAll` executes tasks sequentially, while `Task.WaitAll` executes tasks in parallel.
- d) `Task.WhenAll` cancels all tasks if one fails, while `Task.WaitAll` only cancels the failing task.

Answer: a

Analysis:

- **Option a)** is correct because `Task.WhenAll` is asynchronous and `Task.WaitAll` blocks the calling thread.
- **Option b)** is incorrect because it reverses the behavior of `Task.WhenAll` and `Task.WaitAll`.

- **Option c)** is incorrect because both methods wait for tasks to complete, not control their execution order.
- **Option d)** is incorrect because neither method cancels tasks as described.

9. What is the primary benefit of using `Task.Run` to handle CPU-bound operations?

- a) It helps balance the workload among all available CPU cores evenly.
- b) It ensures the operation runs synchronously.
- c) It prevents the operation from using multiple threads.
- d) It offloads the operation to the thread pool, freeing the main thread.

Answer: d

Analysis:

- **Option a)** is incorrect because `Task.Run` does not guarantee even workload balancing among CPU cores; it primarily offloads work to the thread pool.
- **Option b)** is incorrect because `Task.Run` runs operations asynchronously.
- **Option c)** is incorrect because `Task.Run` can utilize multiple threads from the thread pool.
- **Option d)** is correct because `Task.Run` offloads the CPU-bound operation to the thread pool, which helps free up the main thread for other tasks.

10. How does `Parallel.ForEach` differ from a regular `foreach` loop in C#?

- a) It is always slower than a regular `foreach` loop.
- b) It only works with specific collections and data types.
- c) It allows iterations to run concurrently.
- d) It executes iterations one after the other in a single thread.

Answer: c

Analysis:

- **Option a)** is incorrect because `Parallel.ForEach` can be faster in many scenarios.
- **Option b)** is incorrect because `Parallel.ForEach` works with various collections.
- **Option c)** is correct because `Parallel.ForEach` runs iterations concurrently.
- **Option d)** is incorrect because a regular `foreach` executes sequentially, while `Parallel.ForEach` runs concurrently.

11. Given the following code snippet, what will be the output and how does `await Task.WhenAny` affect the execution?

```csharp
using System;
using System.Threading.Tasks;
```

```
class Program
{
    static async Task Main()
    {
        await RunTasksAsync();
        Console.WriteLine("First task completed.");
    }

    static async Task RunTasksAsync()
    {
        var task1 = Task.Run(() => Task.Delay(1000));
        var task2 = Task.Run(() => Task.Delay(2000));
        await Task.WhenAny(task1, task2);
    }
}
```

- a) "First task completed." is printed immediately.
- b) "First task completed." is printed after 1 second.
- c) "First task completed." is printed after 2 seconds.
- d) An exception is thrown because `Task.WhenAny` cannot be awaited.

Answer: b

Analysis:

- **Option a)** is incorrect because the program waits for one of the tasks to complete.
- **Option b)** is correct because `Task.WhenAny` completes when the first task finishes (after 1 second).
- **Option c)** is incorrect because `Task.WhenAny` does not wait for both tasks to complete.
- **Option d)** is incorrect because `Task.WhenAny` can be awaited properly.