Q1) In one of the calls to Q1() the following are the first six number before sorting:

32 21 76 40 92 17

The following is first six numbers of the same array of integers:

21 32 40 76 17 13

The problem seems to be that the inner while-loop is not checking for all of the previous items in the array, but only checking the item immediately preceding it. To correct the problem, we will need to decrement j. The following is one way of correcting the issue:

while (j > 0 && nums[j] < nums[j - 1])  
{  
 int temp = nums[j];  
  nums[j] = nums[j - 1];  
   nums[--j] = temp;  
}

Q2) The null reference exception is thrown because according to MSDN, the Type.GetMethod(String) method “searches for the public method with the specified name.” Again, according to MSDN, for Classes and Structs “Internal is the default if no access modifier is specified.” Therefore typeof(Q2Class).GetMethod("Function"); is looking for a public method. To correct this, we will just need to specify the access modifier of the “Function” method to be public:

class Q2Class { public int X = 0; public void Function() { X++; } }

Q3)

* Lossless – 4.25 can be represented in binary as 100.01
* Lossless – Since the above was lossless, assigning it to a double is still lossless because a double is able to store a higher precision.
* Lossy – 1.4 cannot be exactly represented in binary. Therefore, it will try to find the closest to that value.
* Lossless – Although f2 was a lossy operation, assigning f2 to d2 will copy all the bits to d2 which will be able hold all the bits to represent the data. Therefore, going from float to double isn’t a lossy operation.

Q4) I believe this is an example of boxing in C#. So the int type, which is a value type, is being converted to a type object. It wraps the value inside a System.Object and stores it on the managed heap (MSDN). Boxing a value type allocates an object instance on the heap and copies the value into the new object. Therefore, changing the variable num does not affect the value that is contained in the o object.

Q5) According to the Internet Engineering Task Force (IETF):

The Request-Line begins with a method token, followed by the

Request-URI and the protocol version, and ending with CRLF. The

elements are separated by SP characters. No CR or LF is allowed

except in the final CRLF sequence.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

The above protocol shows that the tokens are separated by spaces. The problem with the logic in Q5() is that it will determine isValidHTTPMethod = true; even if there is no space after the valid method.

Q6() Under the assumption that overflow is lossy.

* + (Lossy) Adding two Int.MaxValue will cause the result to be larger than an int. Therefore, the number wraps around back to -2.
* – (Lossy) int.MaxValue - int.MinValue = a value which int cannot hold. Therefore, the number wraps around back to -1.
* \* (Lossy) multiplying two int.MaxValue’s will exceed int’s maximum capacity.
* / (Lossy) dividing two ints performs an integer division. Therefore, the fractional part is truncated since it can't be stored in the result type

Q7()

* method1
  + Problem – According to MSDN, returns “**true** if the caller has the required permissions and path contains the name of an existing file; otherwise, **false**. This method also returns **false** if path is **null**, an invalid path, or a zero-length string. If the caller does not have sufficient permissions to read the specified file, no exception is thrown and the method returns **false** regardless of the existence of path.”
  + However, this **best** represents if the file simply exists.
* method2
  + Problem – According to MSDN, there are many exceptions that can be thrown are problems from opening the file. Therefore, if the intention is just to check if the file exists, this may result in showing that the file does not exist, when it actually does.
* method3
  + Problem – Similar to method2, there may be an exceptions thrown even if file exists resulting in determining that the file does not exist.

Q8() According to MSDN, “anonymous types provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first.” Therefore, we will be unable to change the properties of the node once it has been initialized. This causes issues, if we want to change Value or if we want to sort the singly-linked list.

Q9() When the variable s\_sharedValue is being incremented, this is not performed as an atomic action. There is a chance (likely) that both threads will attempt to increment this variable at the same time, which will cause some increments to not perform properly. According to MSDN, “the Interlocked class provides methods that synchronize access to a variable that is shared by multiple threads.” Within this class, there is an Increment method that increments a specified variable and stores the result as an atomic operation.

Q10() According to specifications in XML 1.1, there are a range of characters that are invalid, or is restricted and highly discouraged.

[Definition: A parsed entity contains **text**, a sequence of [characters](https://www.w3.org/TR/2000/REC-xml-20001006#dt-character), which may represent markup or character data.] [Definition: A **character** is an atomic unit of text as specified by ISO/IEC 10646 [[ISO/IEC 10646]](https://www.w3.org/TR/2000/REC-xml-20001006#ISO10646) (see also [[ISO/IEC 10646-2000]](https://www.w3.org/TR/2000/REC-xml-20001006#ISO10646-2000)). Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646. The versions of these standards cited in [**A.1 Normative References**](https://www.w3.org/TR/2000/REC-xml-20001006#sec-existing-stds) were current at the time this document was prepared. New characters may be added to these standards by amendments or new editions. Consequently, XML processors must accept any character in the range specified for [Char](https://www.w3.org/TR/2000/REC-xml-20001006#NT-Char). The use of "compatibility characters", as defined in section 6.8 of [[Unicode]](https://www.w3.org/TR/2000/REC-xml-20001006#Unicode) (see also D21 in section 3.6 of [[Unicode3]](https://www.w3.org/TR/2000/REC-xml-20001006#Unicode3)), is discouraged.]

**Character Range**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [2] | Char | ::= | #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF] | */\* any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. \*/* |

(from https://www.w3.org/TR/2000/REC-xml-20001006#charsets)

Therefore, setting stringValue to a character outside this range may cause it throw an exception. A few of these may be ‘\b’ == 0x08, ‘\a’ == 0x07, ‘\v’ == 0x0B, and ‘\f’ == 0x0C