OOP Sample Question 2 Solution

(*a*)

(*i*) We do not expect to be adding or removing `Photo` objects very often (they will almost certainly be created at startup), so efficient insertion and removal is not a big concern. We may need to search for photos, so efficient search might be useful. This hints at any of the structures that keep their contents in a stable, sorted order. A `TreeSet<Photo>` might be appropriate for example (make sure you identify an implementation such as TreeSet and not just an interface such as SortedSet).

(*ii*)

- Make `Photo` implement `Comparable<Photo>` (I would expect the use of Generics) (1 mark)
- Declare an appropriate `int compareTo(Photo p)` function (1 mark)
- Provide an implementation for `compareTo()` that uses the `getFileSize()` function to work out whether to return 1, 0, or -1. (3 marks)

(*iii*) The `Comparable` approach allows us to specify a single, natural ordering for `Photo` objects. When there are multiple orderings, we must use `Comparator`s instead. We would write a separate class implementing `Comparator` for each ordering. We would then manually sort the collection using the appropriate `Comparator` class.

(*b*)

(*i*) This is an example of using the *virtual proxy* pattern (1 mark). I would expect to see a UML class diagram similar to Section 4.9 in the notes. You will need to create a `PhotoProxy` class that inherits from `Photo` (2 arks). A subtle difference from the standard usage is that the proxy object will actually *create* a full `Photo` object when the function to get the full image data is called, and then pass the request onto that object. You need to write some pseudocode to explain this (3 marks).

(*iii*) To reclaim the memory allocated to a full `Photo` object, we set all references to it to `null` (1 mark). This means the garbage collector will know it can delete the memory (1 mark). However, Java does not guarantee *when* or even *if* that will happen. So the possible outcomes are that the memory is deleted some time after the references are nullified, or the memory is never deleted (1 mark).