Array Demo 1:

1. The triangular brackets indicate a template. In this case, the array is a templated collection type, able to be used with a variety of different datatypes. Here, the type and size are declared within the brackets.
2. The 'std::' prefix is not necessary in this case due to line 55, in which the namespace std is used. This is supposedly bad practice as it can cause problems should anything share the same name as something from the standard library. Though I have seen arguments for and against. Personally, I'm not against it. It means that I do not have to add the prefix before anything contained within the standard library and so far, I haven't stumbled into any of the issues that others have.
3. They indicate the element type for the collection and the size, respectively.
4. Itr2 is an iterator, just declared in a different way.
5. V is reference to an int.
6. For each element in the array a1, do this:
7. Neither technically work. Square brackets brings up an exception, at() brings up an unhandled exception. Square brackets don't have a range check, whereas at() does.
8. V is an iterator for an array of ints.
9. V is a reference to an int again.
10. `sort(a1.begin(), a1.end());`

Remainder of Tasks:

2. That line initialises a4 using a1. The a1 array is passed through a4's constructor and a copy is made.
3. Nice demo.
4. Adding: push(), removing: pop()
5. Only the top element is accessible. To access other elements, you need to remove the top element first.
6. Queue pop removes the first element placed in, rather than the last element placed in.
7. No, not really. Element access can be front or back. An iterator would need to be used to find a specific element.
8. Yes, though there are few reasons I can think of. A list is going to take a constant time to add or remove elements and there is no default size.
9. Nope, they were different. The current size of the vector is the number of elements that are currently in it, the max size, is the maximum number of elements it can have.
10. The second of the 2, as there were parameters in use.
11. Yes, the deconstructor was called due to the Particles being out of scope once the function had completed.
12. Yes, this is due to no "new" keyword being used. Similarly to how the elements were deconstructed, so was this.
13. Emplace_back is the newer C++11 equivalent. Emplace_back is able to reallocate memory to the vector, should the capacity be smaller than the new size.