Q.1.1 [line 164] What do the < and > mean or indicate?
The angular brackets indicate a template/blueprint for a function or class. It allows them to work with many generic data types without being rewritten for each one (like how a normal blueprint works).

Q.1.2 [line 165] Why don't we need to write std:array here? (Is this good?)
This is fine so long as you have `using namespace std;` at the beginning of the file

Q.1.3 [line 166] Explain what the int and 3 indicate in this case?
We're initializing an int array with length of 3.

Q.1.4 [line 204] In the code above, what is the type of itr2?
It's an array iterator of an int array with length 3

```
std::_Array_iterator<int,3>
```

Q.1.5 [line 211] In the code above, what is the type of v?
v is a reference to a1's elements.

Q.1.6 [line 212] In the code above, what does the & mean in (auto &v : a1)
That makes v a reference to a1's elements.

Q.1.7 [line 220] Try this. Why does a1[3] work but at(3) does not?
Both don't work . . .

Q.1.8 [line 233] auto is awesome. What is the actual type of v that it works out for us?
An array iterator.

Q.1.9 [line 240] auto is still awesome. What is the actual type of v here?
A reference to a1's elements.

Q.1.10 [line 250] How would you do a forward (not reverse) sort?

```
sort(a1.rbegin(), a1.rend(), std::greater<>());
```

https://www.techiedelight.com/sort-vector-descending-order-cpp

Q.2 [line 105] In array_demo_2, explain what a4(a1) does
It seems to be the quicker way of declaring a variable that has the same values as a different variable

Q.3 [line 108] No questions for array_demo_3, it's just a demo of Struct/Class use with array.

Q.4 [line 111] How do we (what methods) add and remove items to a stack?
Add = push(), Remove = pop()

Q.5 [line 112] A stack has no no [] or at() method - why?
A stack is LIFO, so there's not much you can do. It's a restricted list.

Q.6 [line 115] What is the difference between a stack.pop() and a queue.pop() ?
A stack is LIFO, but a queue is FIFO, so stack.pop() removes the top element, whereas queue.pop() removes the oldest element in the queue.

Q.7 [line 118] Can we access a list value using and int index? Explain.
No. If we do it'd become an O(n^2) operation, which is bad.

Q.8 [line 119] Is there a reason to use a list instead of a vector?
You'd wanna use a list if:
- You want the ability to insert elements into any spot in the collection efficiently
- You have little need for iterating through the collection one by one to get the one specific element that you need
- You wanna use iterator on the collection without worrying about iterator invalidation
- You need list's specific functions like sorting, splicing, removing, etc

Q.9 [line 122] Was max_size and size the same? (Can they be different?)
size returns the number of elements ***currently*** in the vector.
max_size returns the maximum number of elements the vector can hold.

Q.10 [line 123] Which ParticleClass constructor was called?
ParticleClass(x, y)

Q.11 [line 124] Were the ParticleClass instances deleted? If so, how?
Yes, assuming you're asking after the if loop, because this exists on the stack

Q.12 [line 125] Was the vector instance deleted? If so, how do you know this?
Yes, assuming you're asking after the if loop, because this exists on the stack

Q.13 [line 126] Your IDE might suggest to use emplace_back instead of push_back. What does this mean?
push_back takes an object, whereas emplace_back only takes the values of the new object. With using push_back, it needs to create a new element, then add it to container, then copy the values into there. But with emplace_back, it doesn't copy, it creates the new element using the values. This saves you one step.