1. **Introduce an integrated solution using Abstract Factory Pattern for the above descriptions and Explain the advantages and trade-off of each folder in terms of SOLID principles, cohesion and coupling.**

S - Single Responsibility Principle :
except the Single Responsibility in the Master Control file as the Master Control file is modified to call four different behaviours(input,sorter,shifter) except that we can see that the code modules have only one responsibility to perform.
We can see that every module or class have only one responsibility to perform if it is taking input then only input behaviour from that class is reflected.

O - Open close Principle :
Every module the code need not to be modified or changed as per behaviour requirements rather we can extend the code to reach the desired functionality. As we can observe from the code that if we need to add any more functionality, we need to extend the code rather than commenting out or modifying the code.

L - Liskov substitution principle:
We can use either the derived class or the base class for the code to call the base class objects without the need of any modifications.

I - Interface segregation principle:
There is no model that forcefully is required to use the extra implemented behaviours from the interface or the abstract classes.

D - Dependency inversion principle:
As we can see from code, we can see that everything is hidden and dependant on the interfaces and abstract classes rather than other modules.

Advantages:

1 - Using Abstract factory Pattern we have the following advantages :
A : Loose Coupling between modules.
B : Easy to create objects.
C : Use of Interfaces thus leading to good information hiding and encapsulation.

Disadvantages :

1 - Abstract Factory Pattern we can see that initial code was quite easy to understand but as we have applied the both pattern we can see that the code is quite complex now, with additional interfaces and abstract classes it become quote difficult to understand the flow of the data.
2 - As the whole code utilizes the Abstract factory pattern there is a class and interface overload as we need to create new abstract classes and interfaces for every factory to receive or extract data from.

Cohesion and Coupling:

1 - Abstract Factory pattern increases the coupling between the modules to the formation of concrete factories to call out the objects.
2 - There is low Cohesion as in abstract factory pattern we are using concrete factory and factory for the method declaratiion and implementation.

3 - An interface for building families of connected or dependent objects without identifying their concrete classes is provided by the abstract factory design pattern. The factory makes sure that everything produced is connected to everything else and functions as a whole to other modules.

Compare (1) and (2) and mention the advantages and trade-off between them:

1 - For the first pattern we are using the abstract factory pattern , builder pattern and Singleton pattern , for which we can say that to understand the dataflow between the objects and classes becomes quite difficult but as on other hand we can observe that the creation of the complex objects through builder pattern makes the information hiding and encapsulation implemented on the code. There is also less coupling when we use builder and singleton design pattern as compared top only using abstract factory pattern.

- In comparison to using merely the Abstract Factory design, employing Builder and Abstract Factory together can provide more flexibility, better readability, less coupling, and better separation of responsibilities.
- Code can be organized in a way that makes it easier to comprehend by combining the Builder and Abstract Factory patterns. The Builder pattern clearly separates the creation of complicated objects from their representation, whereas the Abstract Factory design clearly separates the client code from the implementation code.

2 - For the second pattern we are only using the Abstract Factory Pattern by replacing the Prototype Pattern, we can see that the data flow between the modules is quite easy to understand as compared to the above patterns but there is huge amount on interdependency between the modules which leads to increase in coupling.

- It also increases the code bloat and makes it hard to maintain.