# Data Structures and Algorithm Design

## Section 1: Outline

A bankbook offers us a physical list of transactions recorded on a deposit account at a bank, for example, a chequing or savings account. A bankbook entry generally provides the date and time of the transaction, a short description, credits, debits, and the closing balance after the transaction. For a digital bankbook, a unique identifier assigned to each transaction would also be ideal. For our purposes, we will generate a unique identifier using the current date and time represented as a Unix timestamp and append an arbitrary number of randomly generated hexadecimal characters so many transactions created at the same time won’t have the same unique identifier. A Unix timestamp represents the number of seconds that have passed since the Unix epoch: January 1, 1970, 00:00:00 UTC. These transaction properties combine to form what is known as a schema, which is a set of properties that the transaction can (or must) include, depending on how we determine what is required for a transaction. For our schema, we will require all properties but we could also create other properties, like a property to store the last time a transaction was updated which would not be required.

## Section 2: Algorithm Design

To interact with the bankbook, a set of algorithms to read and modify the transactions inside will have to be defined. We will need to create new transactions, read existing transactions, and update or delete transactions in the case of an error. In addition to these four primary operations, we will also create a sort operation that will allow us to display our sorted data by property. The four primary operations are commonly referred to as “CRUD” for create, read, update, and delete. For this application, we can assume that we are using a language that has arrays or lists with common methods like insert, find by X, and remove that will eloquently manage the behind-the-scenes logic for things like recreation or reindexing of arrays or lists on inserts or removals. We can also assume that a schema can be easily enforced on elements in an array. If we were writing a real program instead of just pseudocode, this would generally be done either by the database you are using or by writing additional logic. In the following pages, we will define and describe via flowcharts our four primary algorithms along with a sort algorithm with which to operate our bankbook.

To add transactions to our bankbook, let us define a “create” operation. This will include all required information according to our schema, and generate a unique identifier, or UID, as part of the process. In real-world databases, the UID can be called a primary key, unique identifier, object identifier, or similar.

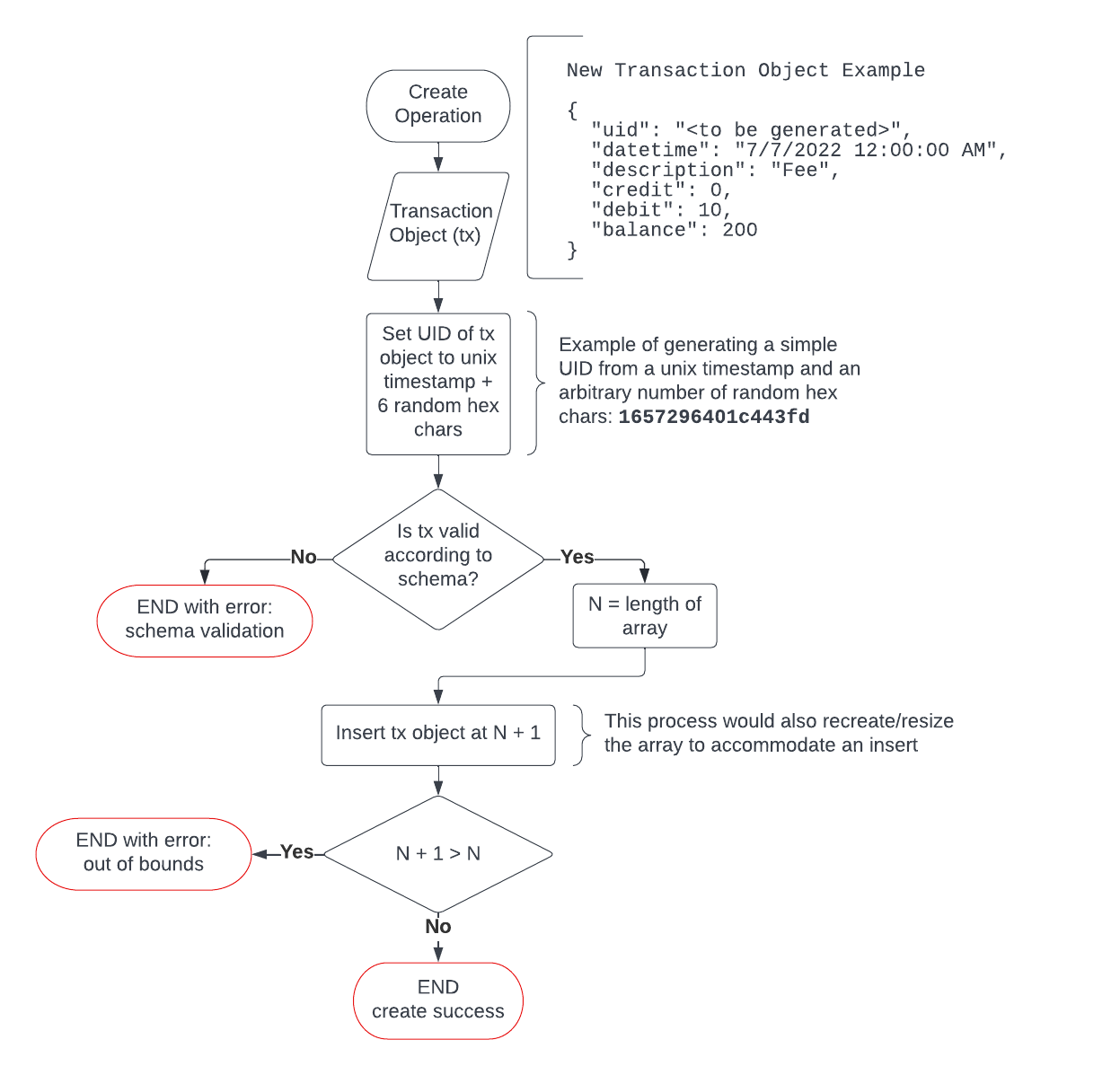


Figure : Create operation

Now that we can create transactions and have them added to our database, we need to be able to read them. For our read operation, we will define an algorithm that can find a transaction object based on its UID. As mentioned previously, we can assume the language we are using has a “find by X” method defined for its arrays.

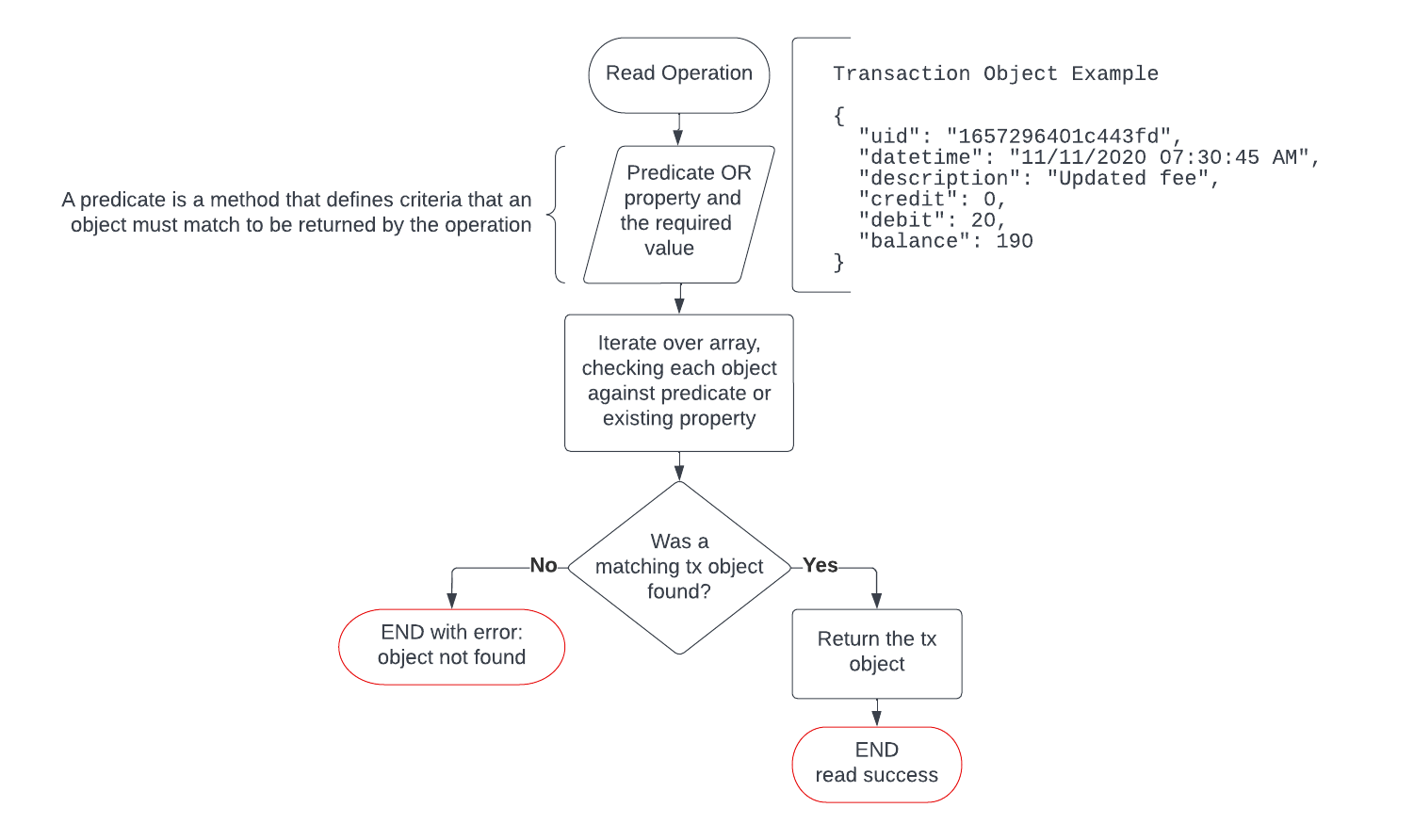


Figure : Read operation

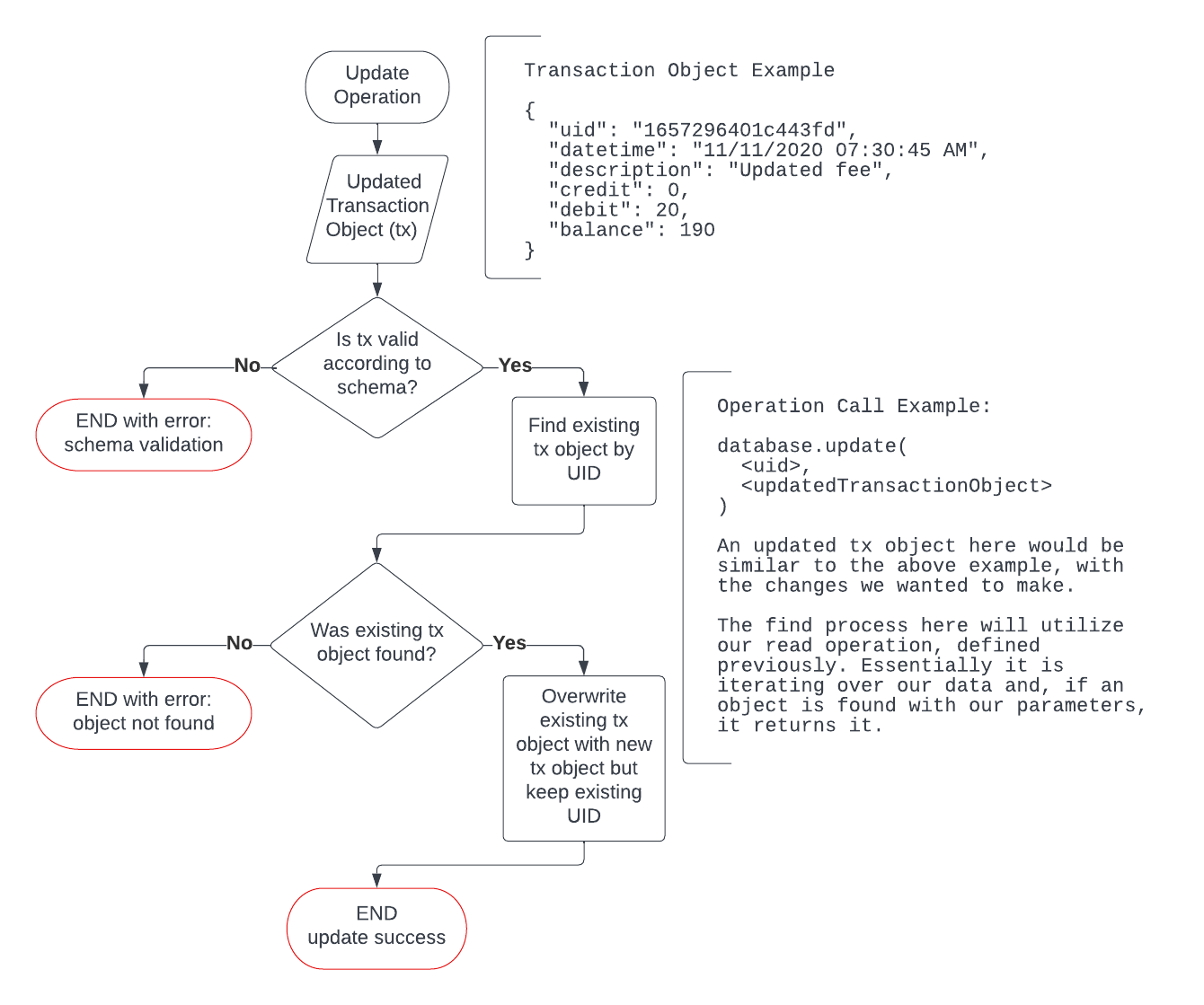
We now have two of four primary functions. If an error is made on a transaction and we would like to fix the error, we will need to define an update operation to overwrite existing transactions with a new transaction object. Ideally, this would include well-defined checks to ensure that the UID was not overwritten, but for the sake of brevity, it has been merged with the overwrite process.

Figure : Update operation

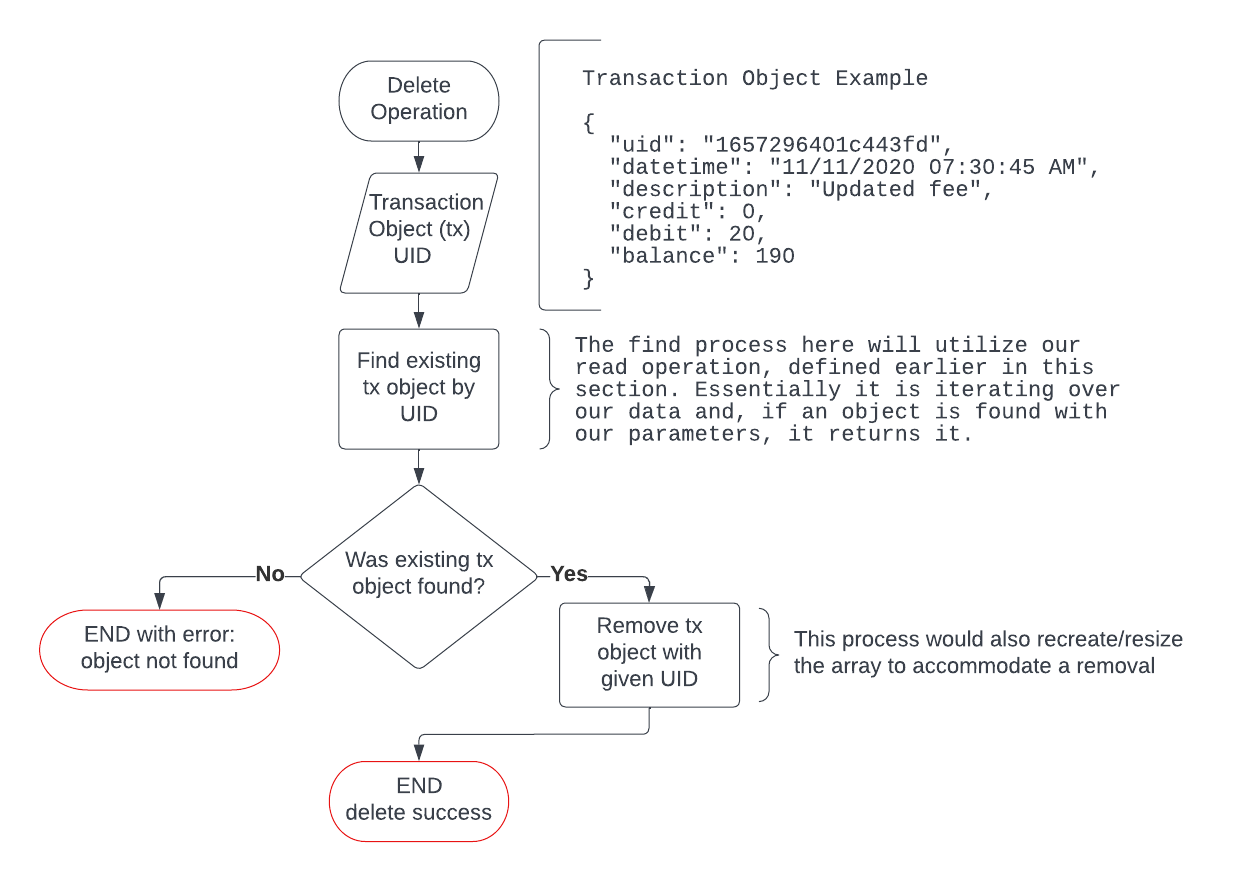
The final primary function for our bankbook application is a delete operation. If an errant transaction is added to our bankbook, we should have the ability to remove it entirely. Without going into philosophical, moral, and legal quandaries surrounding data retention and use, we will assume we can simply delete our data without external repercussions.

Figure : Delete operation

Our quality-of-life sort operation is a bit more complex than our primary operations. This particular sorting algorithm uses the simplest sorting method, the bubble sort. The bubble sorting method works by iterating over and swapping adjacent elements in an array until all of the elements are sorted, then returns the sorted array:

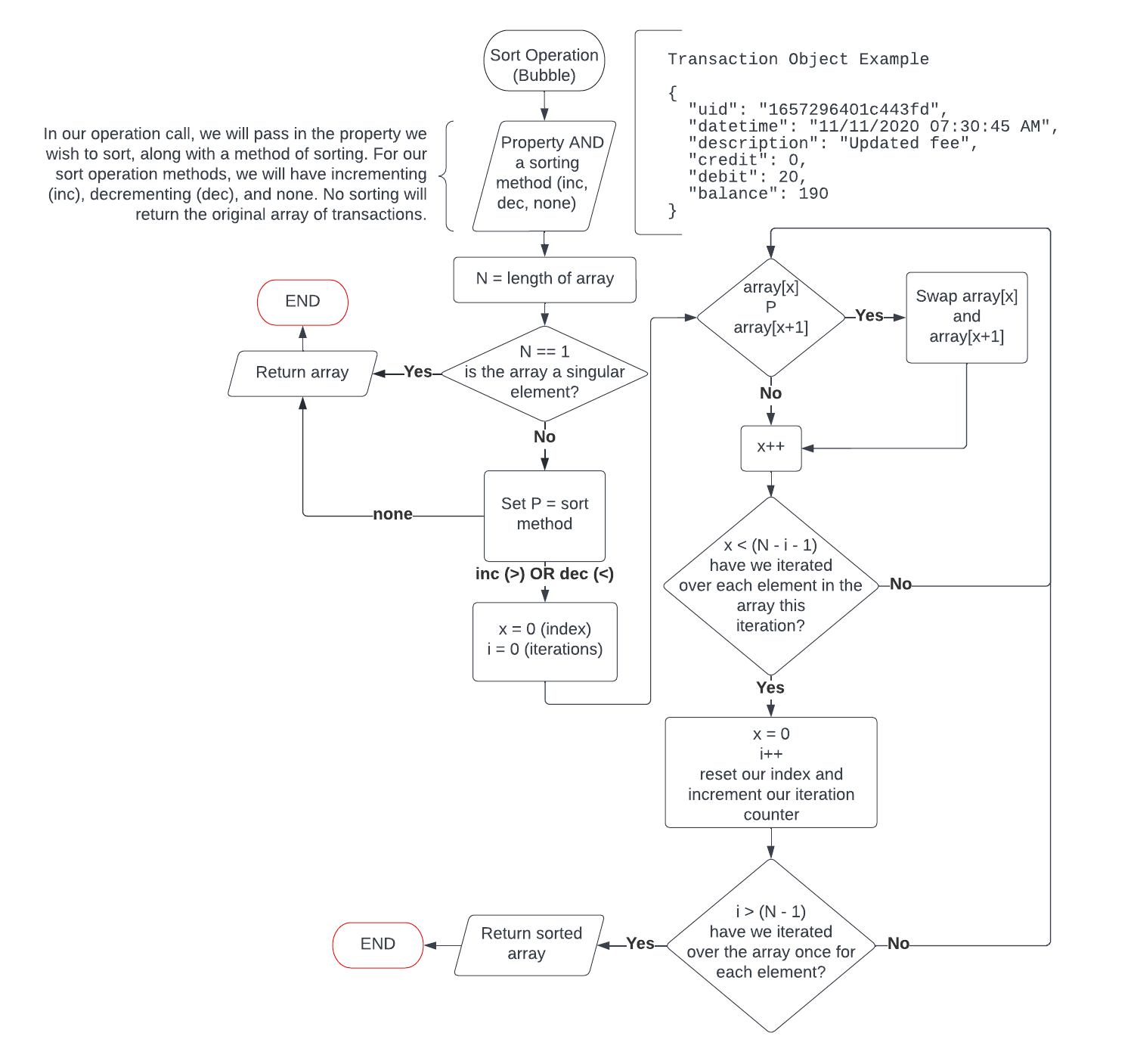


Figure : Sort operation, using the bubble sort method

## Section 3: Testing

Now that we have our primary operation algorithms written, we can interact with our bankbook application. See figures 6 and 7 below for the test plan grid, including our expected results from our tests. Each test shows the operation used, the test data and the operation call, and the expected result.



Figure : Testing table

## Section 3: Testing (continued)



Figure : Testing table (continued)

## Section 4: Conclusion

In conclusion, we have identified and defined the four primary CRUD operations we would need to operate a digital bankbook. In addition, we defined a sort algorithm to help us efficiently display the information we need. If we wanted to expand our feature set, we could write additional sorting algorithms or a find operation that returned multiple transactions at once. What we are left with is a simple but robust digital bankbook to record and manage our banking transactions.