Q1. Case Study: ATM Finder for Mark with Cash Availability Status.

Mark is a resident of Downtown, Salt Lake City, Utah. He is developing a program to help his friends find the nearest ATM in the given area with cash availability. Mark is looking for programmers like you to put in his team.

You are tasked with developing a program to find the nearest ATM (Automated Teller Machine) for users in a given area. The program should utilize basic data structures such as trees, linked lists, and stacks to efficiently locate the nearest ATM. Additionally, you need to consider whether the ATM has cash available.

Consider the following scenario:

You are provided with a list of ATM locations, each containing its Cartesian coordinates (x, y), the bank it belongs to, and a flag indicating whether the ATM has cash available or not. Users will input their current location (x, y), and your program should output the nearest ATM location that has cash available, along with the bank it belongs to.

To solve this problem, you can employ the following approach:

1. Organize the ATM locations using a tree structure, where each node represents an ATM location.

2. Implement a method to calculate the Euclidean distance between two sets of Cartesian coordinates (x1, y1) and (x2, y2).

3. Use a stack or a linked list to traverse the tree and find the nearest ATM location based on the user's input coordinates, considering cash availability.

Here are 8 questions answering which in sequence will lead you to the complete logic for Mark's problem. Remember skipping a step will lead you to incomplete solutions.

1. Write a Python class definition for an ATM location, including attributes for Cartesian coordinates (x, y), bank name, and a flag for cash availability.

2. Implement a function in Python to construct a binary search tree from a list of ATM locations, considering cash availability.

3. Write a Python function to calculate the Euclidean distance between two sets of Cartesian coordinates (x1, y1) and (x2, y2).

4. Develop a Python function to traverse the binary search tree and find the nearest ATM location to a given set of coordinates, considering cash availability.

5. Choose between using a stack or a linked list for tree traversal in your Python program.

6. Describe the Python algorithm to traverse the binary search tree and find the nearest ATM location using the chosen data structure, considering cash availability.

7. Modify your Python code to handle cases where there are multiple ATM locations with cash available at the same distance from the user's coordinates. (Higher cash amount must be preferred)

8. Implement error handling in your Python program to deal with edge cases such as invalid user input or an empty list of ATM locations.

```
ATM_locations = [
  [53.78654321049217, -77.14530986472028, 'Bank B', False, 2613],
  [74.27853809187917, 56.933595727051585, 'Bank D', True, 8505],
  [-62.04300134255429, 1.5121462919326566, 'Bank C', True, 7314],
  [-68.44011130411411, -86.21194687528535, 'Bank A', True, 2902],
  [24.503174340368455, -28.948000022190422, 'Bank A', True, 1389],
  [49.07332786051396, 69.24477363521358, 'Bank D', False, 8748],
  [-69.94475756415214, 11.407926764525037, 'Bank D', True, 5503],
  [-99.471289476157, 44.935746698810085, 'Bank D', True, 1953],
  [61.667436585734195, -69.73260921411032, 'Bank A', False, 2543],
  [-5.732788030823154, 86.79344992443498, 'Bank A', True, 6946],
  [23.188746775981633, -14.648488073221716, 'Bank D', True, 8548],
  [58.1155683709083, -52.38933405466107, 'Bank A', True, 1681],
  [-81.87258950769083, 94.248590190033, 'Bank D', True, 3122],
  [42.33611364994041, -20.998256115589832, 'Bank D', False, 8532],
  [-78.13888133908836, -17.684644739416166, 'Bank A', False, 3134],
  [13.42296333487475, -85.1594022486322, 'Bank B', False, 2089],
  [-54.065210837539616, -5.827220614126881, 'Bank C', True, 3476],
  [96.86261546534437, 33.7857227586493, 'Bank B', False, 8222],
  [94.58146605378892, 70.83762435687422, 'Bank C', True, 1094],
  [88.21674042854578, -33.67505117151755, 'Bank B', False, 4995]
]
```

Q2. Case Study: Online Bookstore Inventory Management

You have been tasked with developing a simple inventory management system for an online bookstore. The system should allow the user to perform various operations such as adding new books, updating book information, searching for books, and generating reports.

Requirements:

1. Implement a menu-based interface allowing the user to choose from the following options:
   - Add a new book to the inventory.
   - Update the information of an existing book.
   - Search for a book by title.
   - Display all books in the inventory.
   - Generate a report showing the total number of books in each category.

2. Each book in the inventory should have the following attributes:
   - Title
   - Author
   - ISBN (International Standard Book Number) (Any 10 digit number will work)
   - Category (e.g., Fiction, Non-fiction, Science Fiction, Mystery, etc.)
   - Price
   - Quantity available in stock

3. Use appropriate data structures such as lists, dictionaries, and tuples to store and manage the inventory data.

Instructions:

1. Implement the necessary functions to fulfill the requirements mentioned above.

2. Create a sample inventory with at least 10 books to demonstrate the functionality of your program.

3. Ensure that the program handles user input errors gracefully, providing helpful error messages when necessary.

4. Try to make the program as fast as possible and error handling must be implemented using try catch block.