**Question No.1**

1. The code uses a list data structure and it is suitable for this because it allows easy access to the elements of the sequence and it can easily be modified to add or remove elements.
2. The purpose of initializing maxSum with the first element of the list is to ensure that the algorithm always returns a valid maximum subarray sum because if the list is empty then the maximum subarray sum is zero which is the value of the first element. Otherwise the maximum subarray sum must be at least as large as the first element.
3. The for loop iterates through the list in a sequential order from the first element to the last.

The sumz variable is used to store the current sum of the elements in the list till the current index. It is continously updated within the loop by adding the current element to the previously stored sum.

1. The code identifies the maximum subarray sum by comparing the current sum of the elements in the list till the current index to the maximum subarray sum obtained at that moment. If the current sum is greater than the maximum subarray sum obtained at that moment, then the maximum subarray sum is updated to the current sum.

This is the condition that triggers the update of the maxSum: **elif maxSum <sumz:**

1. The time complexity of the code for finding the maximum subarray sum is O(n) , where n is the number of elements in the list because the for loop iterates only once through the list.

The code for finding the maximum subarray sum is efficient and performs well because the use of a list data structure allows easy access to the elements of the sequence since they are going to be accessed multiple times. In addition, the algorithm of the for loop is efficient because it only needs to iterate through the list once hence its effectiveness and good performance.