* 1. Lists
* Lists are suitable here because they allow efficient random access, preserve order during iteration, and provide helpful built-in functions which are needed to efficiently calculate the running the data

* 1. By initializing maxSum with the first element, it ensures that maxSum always has a valid value to compare other running sums against. Without this initialization, maxSum would be uninitialized and comparing against it in the first iteration of the loop could lead to incorrect results.
  2. The for loop iterates through the list elements from index 0 to the length of the list and on each iteration, it takes the element at the current index using lst[i]. The variable sumz keeps a running total of the sum of elements seen so far in the current sub-array. On the first iteration, sumz is initialized to 0. On the next iterations, the current element lst[i] is added to sumz using sumz += lst[i],this updates sumz to include the running sum of elements from index 0 to the current index i

After each addition, sumz now reflects the sub array sum from the start of the list to element at index i. The purpose of sumz is to calculate this running sum on each iteration to identify the maximum sum sub-array. By continually updating sumz in the loop, after the loop finishes, sumz will be equal to the sum of all elements

* 1. The code identifies the maximum sub-array sum by tracking the running sumz on each iteration and updating maxSum if a new maximum is found.

MaxSum triggers an update when the current iteration's sumz exceeds the previous maximum stored in maxSum which indicates that a new maximum sub array has been found

MaxSum is then updated to track this new maximum. This allows the code to efficiently identify and track the true maximum sub array sum in one pass over the list.

* 1. The time complexity is O(n)

The choice of data structure which is list) and algorithm contributes to Its performance and efficiency through allowing direct access and single pass traversal due to list, Small O(1) operations inside loop, No data stored inside the other or extra trips over data and MAX calculation reuses sub problem solutions

This results in an overall linear O(n) time complexity.

The choice of list DS and this maximum sub array algorithm work very efficiently together due to the properties above. It achieves optimal linear time performance for this problem.

**OBJECTIVES**

**ANSWERS.**

1. **C** .append()

2. **B** .my\_list.insert(0,’x’)

3. **B**. [111,7,2,1,4]

4. **D.** All the above

5. **C**. my\_list.pop(0)

6. **C**. makes|1 and |2 point to the same list

7. **A**. ‘apple’ not in fruits

8. **B** . (3,4,5)

9. **A**. My\_list[-2]

10. .**A** .true