Q1

1) A user needs to maintain a database that stores individuals' names along with their blood types (O+, O-, A+, A-, B+, B-, AB+, AB-). The database must support the following requirements:
    a) The strength of AUList is the efficient insertions. Since the list is unsorted adding to a new record takes the same amount of time and can take the next available position.
    b) Since the list is unsorted the searching method happens linearly so it can be slow if what you need is further down the list
    c) I would recommend the ASList structure since searching can be name based and takes little to no time to find what you need. Plus, you already know the maxium number of records needed.
2) A user needs to manage a list of up to 100 favorite songs with the following requirements:
    a) LLUList has efficient insertions so adding songs is not a problem due to no sorting
    b) The biggest weakness is having to clear the list by deleting each individual node and that can take a long time for big lists.
    c) I would suggest the AUList since there is no need for sorting or searching. You can insert new songs by putting them in the next available node. And clearing the list can be done in a single command by setting the array length to 0
3) A user needs to store information about recent sightings of a specific animal (such as the sighting location) with the following requirements:

a) LLSList does allow for efficient retrieval of elements in a certain order. Which in this case would be date or location
b) The weakness here is the slow insertions of new sightings.
c) I would recommend LLUList because it has fast insertions, the unknown list length is not a problem since the linked list dynamically adjusts, and you get efficient retrieval of new sightings since the newest are added to the front of the list.

Q2

1) Linear Search vs. Binary Search – Define both search algorithms. Compare their efficiency using Big-O notation. Discuss when each method is more appropriate based on the data being searched.
    a) Linear Search: Best for small or unsorted datasets. It is also ideal when the list is constantly changing, as it does not require pre-sorting. It is useful when simplicity is more important than performance.
    b) Binary Search: Ideal for large, sorted datasets. It offers faster search times when the list does not change often. Binary search is well-suited for scenarios where fast lookups are needed, such as in databases. However, it is not suitable for unsorted lists, as the data must be sorted first.
2) Stack ADT vs. Queue ADT – Explain the principles of both abstract data types (ADTs) and how they manage data. Compare their similarities and differences. Discuss which one is easier to implement efficiently and why.
    a) Stacks are easier to implement efficiently because they only require managing a single pointer (top of the stack) and perform both push and pop operations in

constant time whether implemented using an array or linked list. Because of this simplicity, stacks are generally easier to implement and manage efficiently compared to queues.

b) Queues require extra management for maintaining both the front and rear pointers. If implemented using an array, a queue may require shifting elements unless a circular array is used. A linked-list implementation of a queue avoids shifting but requires tracking both ends.

3) Dynamically Allocated Array vs. Vector Class (C++) – Define both terms in the context of C++. Compare how they handle memory allocation. Compare their ease of use and efficiency.

a) Dynamically allocated array is created using pointers and the new keyword. The array size is defined at runtime, allowing flexibility, but manual memory management is required to avoid memory leaks. The size is fixed once allocated, and resizing involves manually creating a new array, copying elements, and deleting the old array, which can be inefficient and error prone. Can be more efficient in scenarios where the array size is fixed and known in advance, as it avoids the overhead of dynamic resizing and memory management. But it requires explicit memory management, including allocation, resizing, and deallocation.

b) Vector class is a dynamic array that automatically manages memory allocation and resizing, offering dynamic size adjustments, built-in methods for adding/removing elements, and exception safety. When capacity is exceeded, the vector typically doubles its size, reallocating memory and copying existing

elements. Provides high-level functionality through methods like push_back(), pop_back(), size(), and clear(). Exception-safe and easier to use for dynamic data storage.