Question 4: Linked lists & Arrays

Questions:

Compare advantages and disadvantages of arrays vs linked list (complexity of task completion)

```
A large advantage of linked lists is resizability. In many languages, the sizes of
arrays cannot change; however, this is not true in python. Arrays in python are
dynamic, but are inefficient when memory relocation is necessary to increase the
size. Linked lists do not hold this burden; consequently, they are optimal when
the desired size of an array is unknown. Insertion and deletion are also very easy
when dealing with linked lists, with time complexities of O(1), as only the
pointers of the nodes need to be adjusted. Conversely, insertion and deletion in
arrays is computationally expensive, as all of the elements have to be shifted,
resulting in an O(n) complexity.
Arrays are advantageous when the desired size of of an array is known and provide
advantages such as quick access with indexing. Accessing an element in an array
will have a time complexity of O(1), while linked lists will have to sift through
each node, resulting in an O(n) complexity.
```

For arrays, we are interested in implementing a replace function that acts as a deletion followed by insertion. How can this function be implemented to minimize the impact of each of the standalone tasks

```
The replace function (assuming a value to find and replace, and the value to
replace with) could do both the deletion and the insertion in the same line of
code. This is done by looping through each element of the array until we find
the desired value. Once the value is found we can use the iterator variable to
hold our index, then in one line of code we can set the value of the index at the
array equal to the value to replace with. Since python has a garbage collector,
the value that we replaced with will get deleted. This minimizes the impact of
each standalone task efficiently, as the most taxing component of each task is
shifting each element of the array following the insertion/deletion. By simply
changing the value of the array at the target index, this shifting won't occur.
```

Assuming you are tasked to implement a Singly Linked List with a sort function, given the list of sort functions below, state the feasibility of using each one of them and elaborate why is it possible or not to use them. Also show the expected complexity for each and how it differs from applying it to a regular array

1. Selection Sort:

```
1. It is possible to use a selection sort in a singly linked list, with a
complexity of O(n^2). The difference between applying it to a regular array is
that with a regular array the space complexity is O(n) due to an extra copy of the
array being made during the sort (if the array is not dynamic). While a selection
sort with a linked list has O(1) space complexity, due to the sorting being done
in place, and only needing a constant amount of memory.
```

2. Insertion Sort:

```
2. It is possible to use an insertion sort in a singly linked list, with a
complexity of O(n^2). The complexities are the same as a python array, but a
static array would result in a space complexity of O(n) (whereas the linked list
is O(1)).
```

3. Bubble Sort:

```
3. It is possible to use an bubble sort for a singly linked list, with a
complexity of O(n^2). The complexities are the same as a python array, but a
static array would result in a space complexity of O(n).
```

NOTE:

```
The previous three sorts are feasible with small linked lists, but if the linked
list is with a lot of data it is more feasible to use a sort of O(nlog(n))
complexity (merge and quick sorts).
```

4. Merge Sort:

```
4. It is possible to use a merge sort on a singly linked list, with a time
complexity of O(nlog(n)) and a space complexity of O(n). This sort is feasible to
use for a larger linked list, as it will perform more effiecent than any of the
O(n^2) sorting algorithims. A python array will have the same space complexity and
time complexity as a singly linked list.
```

5. Quick Sort:

```
5. It is possible to use a quick sort on a singly linked list, although it will
look different than if it was applied to an array. This is due to a singly linked
list lacking the power to find a value at a given index, and instead must traverse
```

through itself to find the value. The pivot element must be also be chosen differently, as it cannot be selected by simply accessing an index in the linked list (as previously mentioned). The complexity is the same as an array, where O(n^2) for worst case complexity and O(nlog(n)) on average. This sort is feasible for larger linked lists as it will hold an O(nlog(n)) time complexity, performing better than O(n^2) sorting algorithims.