Would this work? (hint: no)

```
SNode *n = new SNode(9);
node4->next = n;
n->next = node4->next;
```

Why not

This would not work because it sets n’s next to itself, which would create a loop within itself, which would stop it from continuing

(2) Try on Paper: a. Define the terms (on page 1)

ADT (Abstract Data type)-

We can have an abstract concept of a data structure, and then implement in more than one way and describes what the data type is and what it should do

List -

The items have an order of a certain size n, can be duplicated, and they are all the same data type

Push -

Pushes values onto the end of the list

Pop -

Pops values off the end of the list

Stack -

The stack is the memory on which the compiler piles each function’s variables and parameters until they go out of scope

Arrays -

Arrays are all one type, it has an order to it, it has a certain size and it can have duplicates

Time Analysis -

The time analysis is the time to complete the task in order to do a task

Linked List-

Linked lists have the data of each node on the list and a pointer going to another data element. It has no fixed size and there isn’t wasted space

Friend -

A friend class allows another class to use their private and protected methods

Kluge

“A workaround or quick-and-dirty solution that is clumsy, inelegant, inefficient, difficult to extend and hard to maintain”-Google

Given the following code, and the following linked list, if you run the method func4 with this linked list, what would be the resulting list?

a->k->b->o->t->a->h->l->v->a->

```
class SNode {
friend class SLL;
	char c; // as opposed to int data;
	SNode *next;
};
void SLL::func4() {
	SNode *tmp = first->next;
	delete first;
	first = tmp;
```

```
        while ((tmp->next != NULL)&&(tmp->next->next != NULL)) {
                Snode *t2 = tmp->next;
                tmp->next = tmp->next->next;
                tmp = tmp->next; delete t2;
        }
        if (tmp->next != NULL) {
         tmp->next = NULL;
         }
        last = tmp;
}
```

This prints out “koala”

c. With the method pop(), you must loop through the entire list each time, even though the linked list class has a pointer to the last node in the list. Why?

For a single linked list, you can traverse the array in reverse, so to get to the node right before the last

d. When inserting into a list in the kth position, why do you loop to position k-1 and not to position k?

You go to the k-1 position so that you are able to set the next of k and k-1 to the correct node. If you go to k, you wont be able to set k-1-> next to k, which would create a break in the SLL

e. Based on the class for a singly linked list, described above, why would writing a method that either reverses the list or traverses the list in reverse order be difficult?

Because each node is only linked in the forward direction, going backwards would be very difficult to make

(3)Try on paper:

a. Why is it only O(1) to find the kth element in an array?

It is O(1) because the array is in order, so the kth element is just k from the index of the first element

b. Why is it O(n) to find out whether x is in the list when implemented either as an array or a linked list?

Because if you are looking for a specific element, we would have to check every element, and this happens for linked list and array