Questions

1. Give an algorithm for reversing all the elements of an array list, which uses a constant amount of additional space (other than the array itself). That is, you are not allowed to use a new array for your solution.

2. Give a $O(N)$ algorithm (pseudocode) for removing the first instance of a given object $e$ in a list, assuming the list is represented as an array list and the size of the list is $N$. That is, give an algorithm for $\text{remove}(e)$. In your answer, you can use methods given in the lecture.

   [ASIDE (added Sept 16): In the lecture, I presented a $\text{remove}(i)$ algorithm which removes the element at index $i$ in the list. In the above question, I’m asking for a $\text{remove}(e)$ algorithm which removes a particular object (if it happens to be there).

   Having two methods with the same name but different argument types is called ”overloading” a method. Its very common in Java.]

3. As discussed in the lectures, an important property of arrays is that they have constant time access. This property follows from the fact that all array slots have the same memory size.

   What about an array of strings, in which the strings have possibly different lengths? Does this contradict the property just mentioned? Does one still have constant time access to an element in an array of strings?
Answers

1. Here I sketch the idea with pseudocode. The main idea is to use a 'swap' method. You should have seen in COMP 202 how to swap the references of two values. Recall that you need to use a tmp variable to do so.

```plaintext
swap(j, k){
    tmp = a[j]
    a[j] = a[k]
    a[k] = tmp
}
```

The algorithm then swaps the first and the last, the second and second last, etc.

```plaintext
reverseArrayList() {
    for (i = 0; i < size/2; i++){
        swap(i, size-1-i)
    }
}
```

If the list has an odd number of elements, then it doesn’t touch the middle one, which is fine. For example, consider the case \( i = 13 \). It swaps 0,1,2,3,4,5 with 12,11,10,9,8,7, respectively, and doesn’t touch 6.

2. The following algorithm loops through the list and examines each element at most once. Hence it takes time proportional to \( N \) in the worst case, and so we say it is \( O(N) \).

```plaintext
remove( e ) {
    i = 0
    found = false
    while ((i < size) and (found == false)){
        if a[i] == e
            found = true
            return remove(i) // this method was discussed in the lecture
        else
            i++ // means i = i + 1
    }
    print("sorry pal, I couldn’t find your e")
}
```

3. The slots in the ”array of strings” don’t contain strings, but rather they contain references to (addresses of) strings. The strings themselves are objects. Each string has a location somewhere in memory. So there is no problem here. (Note: saying that the array slots all have constant size is fine. The slots contain references or addresses, which all use the same number of bits. The number of bits depends on which machine we are talking about.)