In order to keep the union functions in constant time, I tried
recipient->tail->next = donor->head;
recipient->tail = donor->tail;

But for some reason that didn't work (also it wouldn't actually make the donor lists empty). So now my plan is to do something along the lines of  
while(size(donor) != 0)
    insert(recipient, size(recipient), remove(donor,0));

but I'm worried that's not actually constant time, since it loops through the entire donor list. Anyone have some ideas? Am I wrong about this second method being linear?

Your second method is indeed linear. Your first approach is better, but the donor doesn't magically become empty. You have to make it empty (in constant time). Draw pictures!

Forgive me if I mislead you in any way.

In my head, because we are not concerned with memory in this project (leaving nodes floating around in undocumented space) a list is effectively empty when its head pointer is null.

So far as the user is concerned that list doesn't exist.

Because we are using tail pointers, one might simply extend that logic to the tail pointer and now all evidence of this list and its contents is erased from history. These operations should be constant.

If this sounds incorrect, PLEASE reply, as I certainly would need a correction to my own work.

I hope you're not incorrect, because that's what I did. I'm pretty sure Professor Lusth said that we
didn’t have to worry about collecting garbage until a later project.

Subject: Re: Union
Posted by lusth on Mon, 23 Jan 2017 14:22:28 GMT
View Forum Message <> Reply to Message

There’s no garbage to collect for the union operation. Nodes in the donor are transferred en mass to the recipient.

Subject: Re: Union
Posted by bkaaron on Mon, 23 Jan 2017 23:28:04 GMT
View Forum Message <> Reply to Message

Heads up, just realized while testing boundary/unique cases, my code seg-faulted when trying to union an empty list with a non-empty list. (basically a copy->delete rather than append operation)

I don’t know if Lusth will be testing for this sort of oddball case but I went back and shut my blast doors on this one just to be sure.

Subject: Re: Union
Posted by 1337Programmer on Tue, 24 Jan 2017 03:00:38 GMT
View Forum Message <> Reply to Message

Remember you need to update the size of both the recipient and donor after you are done changing the pointers.

Subject: Re: Union
Posted by lusth on Tue, 24 Jan 2017 16:05:51 GMT
View Forum Message <> Reply to Message

bkaaron wrote on Mon, 23 January 2017 17:28
Heads up, just realized while testing boundary/unique cases, my code seg-faulted when trying to union an empty list with a non-empty list. (basically a copy->delete rather than append operation)

It’s all about the boundaries, ’bout the boundaries.

Subject: Re: Union
Posted by jtevans3 on Tue, 24 Jan 2017 19:09:31 GMT
View Forum Message <> Reply to Message
I used the same
recipient->tail->next = donor->head;
recipient->tail = donor->tail;
and then set
donor->head=donor->tail=NULL;

and my code works for two lists. However, if I rebuild the lists and swap the donor and recipient in
my call, I get a seg fault. Any ideas on what would cause this?

Subject: Re: Union
Posted by jarobinson3 on Tue, 24 Jan 2017 19:19:01 GMT
Make sure you update your size. Do you handle the case where one of the lists is empty? What
about if both lists are empty?

Subject: Re: Union
Posted by jtevans3 on Tue, 24 Jan 2017 19:46:39 GMT
Yes, in my tests it handles either the donor or receiver being empty, and I already have
recipient->size += donor->size;
donor->size = 0;

Subject: Re: Union
Posted by jarobinson3 on Wed, 25 Jan 2017 22:11:35 GMT
The only thing I can think of is that you do not initialize your variable. Possibly in your
insert/remove functions?