Data Structures Exam 1 (96%)

Data Structures (Temple University)

1 Easier Stuff

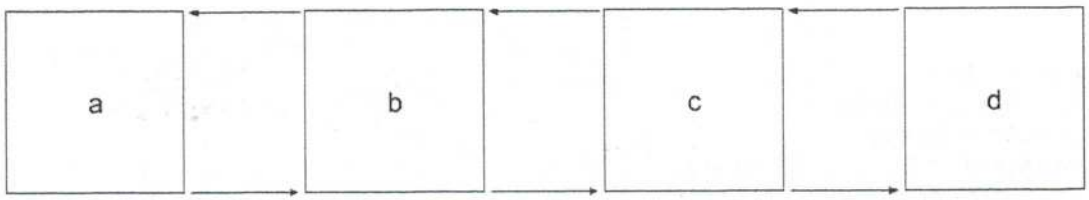


Figure 1: A doubly-linked list. Assume each node has a variable storing their memory location, denoted by the letter on the node.

For each of the following expressions, please indicate which node is being referenced by the chain of variables.

1. (2 points) `a.next.next.next`

1. d

2. (2 points) `c.prev.prev.next`

2. b

3. (2 points) `d.prev.prev.next.prev.next.prev.next`

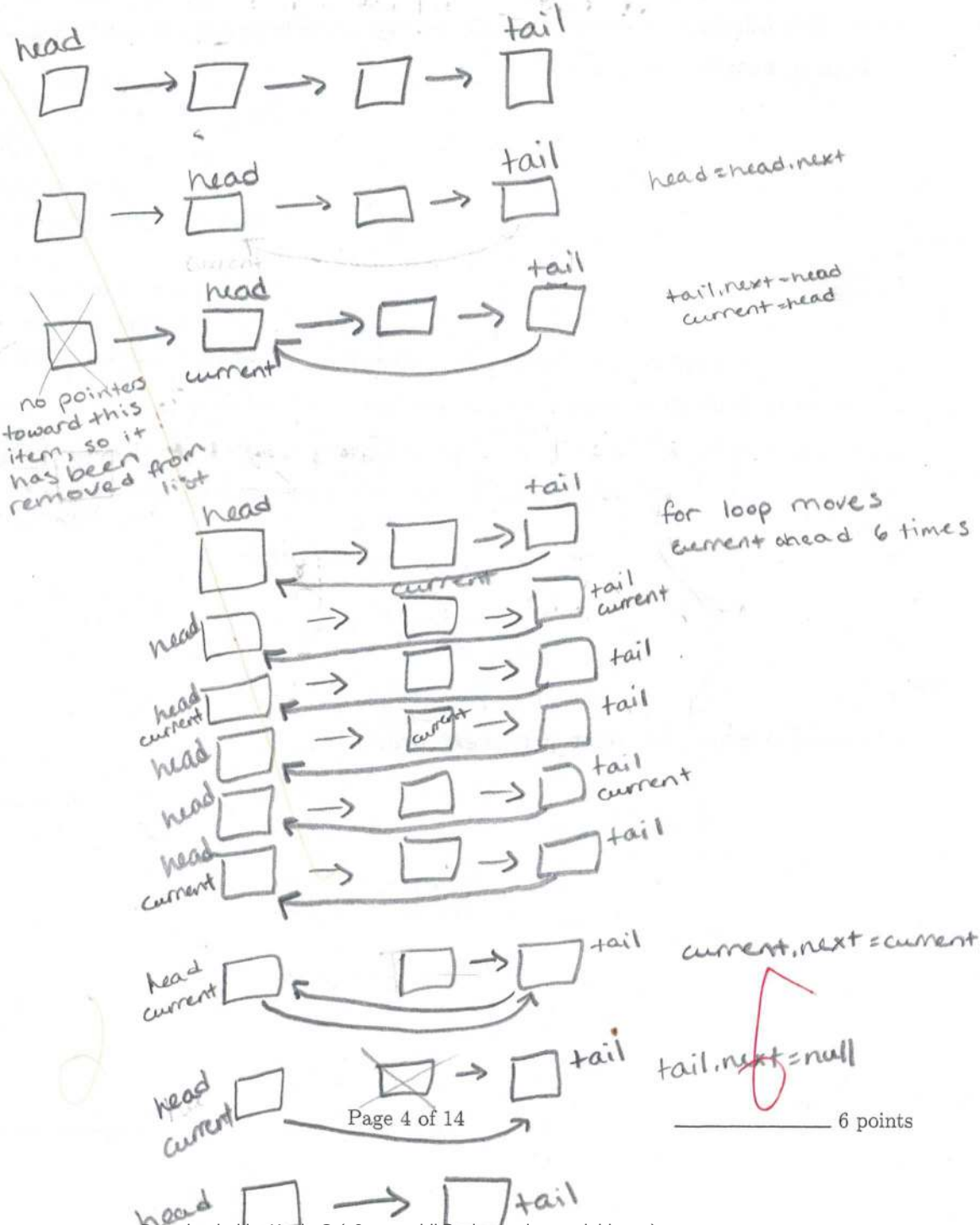
3. c

6

6 points

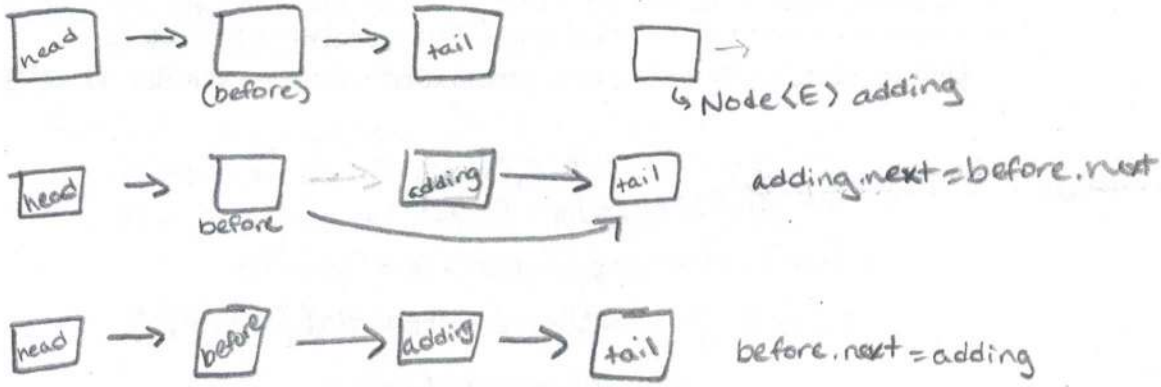
4. (6 points) Draw what happens to a singly-lined `LinkedList` with 4 nodes after the following code is executed. You may draw it step by step.

```
head =  head.next;
tail.next =  head;
current = head;
for(int i = 0; i < 6; i++){
        current = current.next;
}
current.next = current.next.next;
tail.next = null;
```

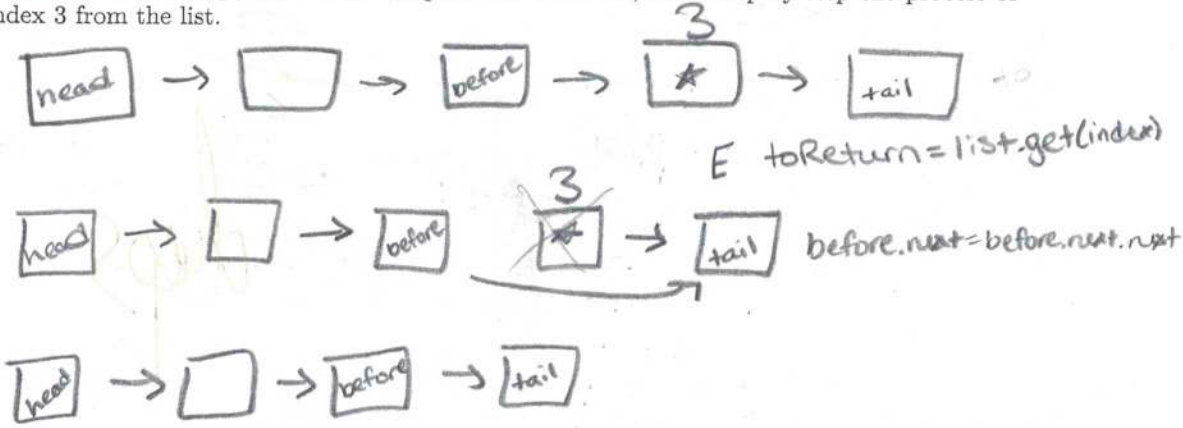


_____ 6 points

5. (5 points) Beginning with a singly linked list composed of 3 nodes, draw step-by-step the process of adding a new item to index 2 of the list. Be sure to note the head at each step.



6. (5 points) Beginning with a singly linked list composed of **5 nodes**, draw step-by-step the process of **removing** index 3 from the list.



10

_____ 10 points

2 List Coding

7. (10 points) Suppose you have some `List` of Strings called `list` and a `String prefix`. Write a method that removes all the Strings from `list` that begin with `prefix`.

```
public static <E> void removePrefixStrings(List<String> list, String prefix){
    String current = "";
    int listSize = list.size();
    for(int i=0; i<listSize-1; i++){
        if (current.startsWith(prefix)){
            list.remove(i);
        }
    }
}
```

2

dups

8. (2 points) What is the time complexity of what you just wrote?

8. ~~2~~ O(n)

8

12 points

9. (10 points) Suppose you have two lists. Write a method `expunge` which will remove all the items in `listB` from `listA`.

```
//Example inputs/changes
// [1,2,3,4,5,5,6] and [2,5,17] --> [1,3,4,6]
public static <E> void expunge(List<E> listA, List<E> listB) {
    for(int i=0; i< listB.size()-1; i++){
        E itemB = listB.get(i);
        for(int j=0; j <listA.size()-1; j++){
            E itemA = listA.get(j);
            if (itemB.equals(itemA)){
                listA.remove(j)
                // remove method will decrease size
                // so the for loop won't go out of bounds
            }
        }
    }
}
```

dups

10. (2 points) What is the time complexity of what you just wrote?

10. $O(n^2)$

2/2

12 points

3 Linked List

Suppose we had a new type of `LinkedList`, called the `SortedLinkedList`, which is a linked list, but it keeps all the items in the list sorted. As a result, when we add an item to a `SortedLinkedList`, we don't provide an index, as the `SortedLinkedList` figures out where to put the new item based on the values already in the list.

Rules:

- You can use either a singly or doubly linked list.
- you can use <, >, and == to compare items, but if you remember how to use `compareTo()`, you can do so for extra credit.
- You may not call the `add(int index, E item)` method.
- You may not call the `remove` method.
- You may not call the `getNode()` method (although you may rewrite it below).

```
private Node<E> getNode(index){
    if(index < 0 || index ≥ size){
        throw new IndexOutOfBoundsException("Index" + index + "is out of
                bounds.");
    }
    Node<E> toReturn = getData(index);
    return toReturn;
}
```

This space intentionally left blank

Wait, I typed something, it's not blank anymore!

_____ 0 points

11. (15 points) Write a method `removeDuplicates()`, which when called by a `SortedLinkedList`, removes all duplicate items from the `SortedLinkedList`. You can assume the `SortedLinkedList` is sorted in increasing order and you don't have to check if the list is empty. You can also assume (for the sake of simplicity) that the `SortedLinkedList` has only numbers.

```
// This is an instance method
// So you have access to head, tail (optional), and the Node class
public void removeDuplicates(){
    int num = 0;
    int temp = 0;
    for(int i=0; i< size-1; i++){
        num = getNode(i);
        for(int j= i+1; j<size-1; j++){
        temp = getNode(j);
        if(num==temp){
            Node<Integer> before = getNode(i-1);
            before.next = before.next.next;
            size--;
        }
    }
}

//getNode is written on page 8
```

12. (2 points) What is the time complexity of this algorithm?

12. $O(n^2)$

2/2

17 points

13. (15 points) Write an method that creates and returns a new `SortedLinkedList` that is the intersection of two `SortedLinkedList` objects. An intersection of two lists contains **only** the items that appear in both lists. Items will only appear once in an intersection.

Additional rules:

- Do not remove any items from the inputs `A` and `B`
- The algorithm must run in $O(n)$ time. This is worth 5 points.

```
// While this is a static method, it's inside of SortedLinkedList
// So you have access to head, tail (optional), and the Node class
// Example inputs/output:
// [1,2,2,3,4,5] and [2,2,2,4] --> [2,4]
// [1,3,5,6] and [1,2,3,5]     --> [1,3,5]
public static SortedLinkedList sortedIntersect(SortedLinkedList A,
                                                SortedLinkedList B){
    List<E> list = new SortedLinkedList<>();
    for(int i=0; i< A.size()-1; i++){
        list.add(A.getNode(i).data); // the instructions didn't
                                     // say I couldn't use
    }                                // add(E item) so I
    for(int j=0; j<B.size()-1; j++)  // didn't want to
        list.add(B.getNode(j).data); // waste time writing
    }                                // out
    for(int k=1; k<list.size(); k++){
        if((list.getNode(k).data).equals(list.getNode(k-1).data)
            Node<E> before = getNode(k);
            before.next = before.next.next;
            size--;
        }
    }
    return list;
}

// getNode is written on page 8
```

8/13

15 points

4 Analysis

14. Please explain how array based lists, like an ArrayList, differ from reference-based lists, such as a LinkedList, in each of the following aspects. Be sure to note where one implementation of a list has a clear advantage over the other. Big **O** notation is recommended.

(a) (5 points) Memory Usage:

Arraylists hold the data stored at each location in the array, whereas LinkedLists also store the references to the next element, which requires significantly more memory usage. However, if the max size of the Arraylist is reached, and it has to reallocate, this will result in a lot more memory usage that will be wasted unless more items are added.

(b) (8 points) Adding and removing items:

In general, it takes O(n) time to add or remove an item fr an Arraylist, because at the front you have to shift n items, or at th middle you have to shift n/2 items. The exception to this is adding the end, which takes O(1) time, except in the rare case that the Arrayli is full, because then we'd need to reallocate, which takes O(n) time. Adding or removing to a location you have reference to, such as the head tail of a LinkedList, takes O(1) time. In all other cases you have to search for a node at the location you're interested in, which takes O(n) tim This makes LinkedLists ideal for stacks and queues, because you're only adding or removing at the head or tail.

(c) (7 points) Accessing an item at a specific index:

Accessing an item at an index takes O(1) time for an Arraylist. However, it takes O(n) time to access an item at an index of a LinkedList, unless you are accessing at the head or tail, which would be O(1). This makes Arraylists ideal for situations where you are accessing items frequently. This makes LinkedLists ideal for situations where you are accessing only the head or tail, such as stacks and queues. 20

 20 points

5 Reverse

15. (5 points) Below you is a method called `reverse`, which reverses the contents of a linked list **in-place**, without the need of creating an extra list. Or rather, the pieces of the method are below you. Put them in the reverse method so that the code will reverse a linked list.

```
head = save;
head = null;
Node<E> save = current;
save.next = head;
Node<E> current = head;
current = current.next;
while (current != null) {
}
```


```
public void reverse() {
    Node<E> current = head;
    head = null;
    while (current != null) {
        Node<E> save = current;
        save.next = head;
        head = save;
        current = current.next;
    }
}
```

Figure 2: How do you do, fellow kids?

5

5 points