19

## 1. Linked Lists Merge (25 pts)

Implement a method to merge two **sorted** linked lists of integers into a single **sorted** linked list *without duplicates*. For example:

l1.data

commons + extras

```
    ptr1
L1: 3->9->12->15->21
L2: 2->3->6->12->19
    ptr2

Merge Result: 2->3->6->9->12->15->19->21
```

Implement a NON-RECURSIVE method to do the merge and return it in a NEW linked list. The original linked lists should not be modified. The new linked list should have a complete new set of Node objects (not shared with the original lists). You may assume that neither of the original lists has any duplicate items.

Your implementation MUST use the efficient $O(m+n)$ algorithm covered in Problem Sets 2 and 3 on a similar problem: using a pointer per list to track simultaneously, traversing each list exactly once. If you use some other inefficient algorithm, you will get AT MOST HALF the credit. You may implement helper methods if needed.

```
public class Node {
   public int data;  public Node next;
   public Node(int data, Node next) {
       this.data = data; this.next = next;
   }
}

// Creates a new linked list consisting of the items that are a union
// of the input sorted lists, in sorted order without duplicates
// Returns the front of the new linked list
public static Node merge(Node frontL1, Node frontL2) {
        // COMPLETE THIS METHOD - YOU MAY ***NOT*** USE RECURSION

if (frontL1 == null && frontL2 == null){
   return null;
   }
 Node  ptr1 = frontL1;
 Node  ptr2 = frontL2;          int Node          + 1
 Node aptr = new Node (null, null);
 Node tmp  = new Node (null, null);
while (ptr1 != null && ptr2 != null){             + 2
 if (ptr1.data < ptr2.data){
    aptr.data = ptr1.data;  ptr1 = ptr1.next;
 }else if(ptr1.data > ptr2.data){
    aptr.data = ptr2.data;  ptr2 = ptr2.next;
 }else {
    aptr.data = ptr1.data; ptr1 = ptr1.next; ptr2 = ptr2.next;
 }
 aptr.next = tmp;

//continue on next page
```

```
if (ptr1 data < ptr2.data) {
    tmp.data = ptr1.data;
    tmp.next = new Node (null, null);
    tmp = tmp.next;
    ptr1 = ptr1.next;
}else if (ptr1.data > ptr2.data) {
    tmp.data = ptr2.data;
    tmp.next = new Node (null, null);
    tmp = tmp.next
    ptr2 = ptr2.next;
}else {
    tmp.data = ptr1.data
    tmp.next = new Node (null, null);
    tmp = tmp.next;
    ptr1 = ptr1.next;
    ptr2 = ptr2.next;
}
return aptr;
}   // end while loop
if (ptr1 == null && ptr2 != null){
    return ptr2; }
if(ptr2 == null && ptr1 != null) {
    return ptr1; }

}
```

+13

+3

13

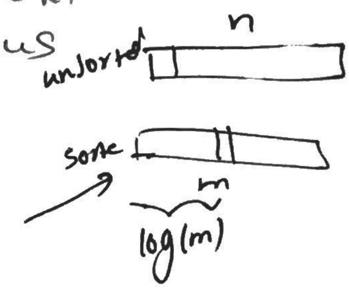
## 2. Array Algorithms Big O (25 pts, 5+7+7+6)

**Compute the big $O$ running time for each of the following.** Briefly describe the algorithm (1-2 sentences), and then give the running time with reasoning. Just putting down a big O answer without algorithm or reasoning will not get any credit.Be concise - if you need more room that is given, you are rambling. Assume that none of the arrays has any duplicate items.

0

a) Worst case big $O$ time of the *fastest algorithm* to print the common elements in two arrays, one unsorted of length $n$ and the other sorted of length $m$. You must work with the original arrays without modifying them. Do not count the time to actually print.

O(m log n). Each element in the sorted array m must be compared with each element after it is sorted in array n. Thus m x log n = O(m log n)




O(n log m)

7

b) Given an array A of integers, worst case big $O$ time of the *fastest algorithm* to compute *partial* sums in a new result array R of the same length. R[i] = sum of integers A[0] through A[i]. So, for instance, if A is [1,5,3,6,2], then R would be [1,6,9,15,17]. (Array A must NOT be modified.)

O(n) Each element needs to be added to the previous element in the resulting array, and because access to each element is equally 1, the run time would be 1 · n. Therefore O(n).

c) Worst case big $O$ time to find and return the 4-th smallest item in an unsorted array of length $n$. Assume $n$ is much larger than 4. Your algorithm can modify the way in which the array items are arranged, and you may also use extra array space if needed. (If you are using a known algorithm in your solution, you may use it's running time without derivation.) *Fastest solution will get full credit, any other will get max 4 points.*

0

It will require log (n) to sort the array. Then, we need to iterate through every element of the array to be sure of the 4th smallest. Thus (n) + log (n) =

$\boxed{O(n)}$ is right

6

d) Big $O$ time of this code, run on an array A of integers of length $n$:

```
for (int i=1,sum=0;i < A.length;i*=2) {
    sum += A[i];
}
```

n = 1 : |

n = 10 : |||

n = 1000 : ~~||||~~ ~~||||~~

n = 100 : ~~||||~~ ||

n = 10,000 : ~~||||~~ ~~||||~~ ||||

O(log n) The worst case for this code would require traversing and an operation on at least a fraction of the entire array, with that fraction get smaller each time.

25

lo hi mid hi 6
0 1 [2] 3 4

lh h
0 1 2 3 4

### 3. Lazy Binary Search (25 pts, 8+7+10)

The following is a *lazy* version of binary search on a sorted array, $A$, of integers:

```
boolean lazySearch(int[] A, int target) {
   int lo=0, hi=A.length-1;
   while (lo < hi) {
      int mid = (lo+hi)/2;
      if (target > A[mid]) {      // C1
         lo = mid + 1;
      } else {
         hi =  mid;
      }
   }
   return target == A[lo];       // C2
}
```

is target > A[mid]

return

0: ||||
1: ||||
2: |||
3: ||
4: |||

For parts (a) and (b), lazy search is done on the following array of integers:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 16 | 26 | 30 | 66 | 76 | 77 | 93 |

a) Show the sequence of > and == comparisons (in the statements marked C1 and C2 respectively in the code) that would be done against the items in the array when searching for 66.

8

mid = 66

Is 66 > 66 ? No , hi = 66

mid = 26

Is 66 > 26 ? yes , low = 26

mid = 30

Is 66 > 30 ? yes low = 66

break while loop

return 66 == 66 // $\boxed{\text{return true}}$

b) Show the sequence of > and == comparisons that would be done against the items in the array when searching for 85.

7

mid = 66
Is 85 > 66? yes , low = 76
mid = 77
Is 85 > 77? yes, low = 93
break while loop
return 85 == 93 $\boxed{\text{return false}}$

c) Given a sorted array of length 5, find the number of comparisons required by the lazy search code above to find a match against items in each of the positions. Count each > as one comparison, and each == as one comparison. Do not count the comparison in the while loop condition. Fill in the second column of the following table with your answers:

10

| Array position | Number of comparisons |
|---|---|
| 0 | 4 |
| 1 | 4 |
| 2 | 3 |
| 3 | 3 |
| 4 | 3 |

work on previous page