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: 3->9->12->15->21
L2: 2->3->6->12->19

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) {
    return frontL2; }

else if ( frontL2 == null) {
    return frontL1; }

Node merge; L1 = frontL1, L2 = frontL2; merge ptr = null;

Node merge = new Node (0, null);

Node ptr = merge;
```

ON BACK

```
while (L1 != null || L2 != null) {
  if (L1.data < L2.data) {
Node temp = new Node (L1.data, null);
    ptr.next = temp;
    ptr = ptr.next;
  L1 = L1.next;          }
else if (L2.data < L1.data) {
  Node temp = new Node(L2.data, null);
      ptr.next = temp;
      ptr = ptr.next;
      L2 = L2.next;          }
```

23

```
else {
    L1 = L1.next;
    L2 = L2.next;
}
```

new node!! -2

```
}
```

```
if (L1 = null) {
  while (L2 != null) {
    Node temp = new Node (L2.data, null);
      ptr.next = temp;
      ptr = ptr.next;
    L2 = L2.next }
else if (L2 = null) {
    while (L1 != null) {
  Node temp = new Node (L2.data, null);
    ptr.next = temp;
     ptr = ptr.next;  }
     L1 = L1.next
```

On BACK ↪

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.

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.

THE ALGORITHM CHECKS EACH OF THE VALUES OF THE UNSORTED ARRAY WITH EACH OF THE VALUES OF THE SORTED ARRAY USING BINARY SEARCH. IF THE COMMON ELEMENT IS FOUND, PRINT, ELSE MOVE ON UNTIL THE UNSOR ARRAY HAS BEEN TRAVERSED.

PARSE UNSORTED ARRAY: $O(n)$

PARSE SORTED ARRAY: $O(\log m)$

5

SORTED ARRAY IS PARSED n TIMES SO IT IS $\boxed{O(n \log m)}$

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.)

CREATE ARRAY R $\{O(1)\}$, THEN PUT FIRST VAL OF R[] AS A[] $\{O(1)\}$

NEXT GO THROUGH ARRAY A AND HAVE ARRAY R SET THE VALUE FOR R[n] TO BE R[n-1] + A[n]. $\{O(n)\}$.

6

ASSUMING WE COUNT BOTH TRAVERSING ARRAY A } INPUTTING A VALUE INTO ARRAY R INTO BIG O, THE RUN TIME WOULD ADD n VALUES INTO R while going THROUGH ALL n INDEXES OF A.

SO IT WOULD BE ~~$O(n^2)$~~.

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.*

TRAVERSE ARRAY AND FIND SMALLEST VALUE USING SEQUENTIAL SEARCH.
TRAVERSE ARRAY AGAIN TO FIND THE 2ND SMALLEST VALUE USING SEQUENTIAL SEARCH
TRAVERSE ARRAY 3RD TIME TO FIND THE 3RD SMALLEST VALUE USING SEQUENTIAL SEARCH
TRAVERSE ARRAY 4TH TIME TO FIND THE NTH SMALLEST VALUE USING SEQUENTIAL SEARCH
EACH TIME YOU FIND THE ONE NUMBER THAT IS SMALLEST AFTER PREVIOUS ONE, SO NOW YOU HAVE 4TH SMALLEST AND CAN RETURN IT.

$O(4n) = O(n)$

5

$(n-1) + (n-2) + \cdots$

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

If len = 10, i goes: 1, 2, 4, 8. sum occurs 4x, so 8 actions.
if len = 50; i goes; 1, 2, 4, 8, 16, 32. sum occurs 6x so 12 actions

$2^x \leq n$

$2^x = 5$

$\log_2 32 = 5$

$\log_2 5 = x$

$\log(n) \times 2$ (for count 1 & for sum)

$O(\log n)$

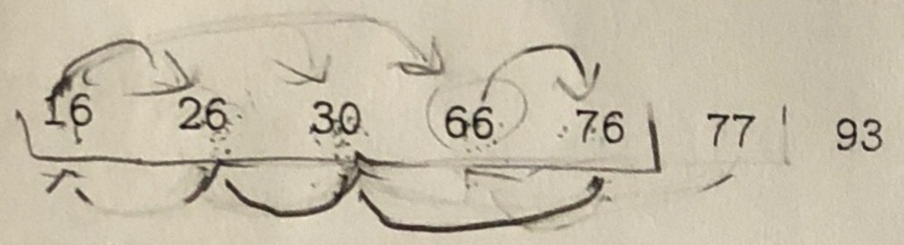
7

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
}
```

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



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.

16 26 30 66 76 77 93
LO MED HI

LO=6 30 MID=3;

C1

16 26 30 66 76 77 93
LO MED HI

3/2
MED=1 C1

16 26 30 66 76 77 93
LO HI

MED = LOW = 2

C1

16 26 30 66 76 77 93
LO, HI

BREAK

C2

3 C1 STATEMENTS AND THEN THE RETURN STATEMENT C2 IN THE END.

+8

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

16 26 30 66 76 77 93
Lo Hi

THERE ARE 2 C7 COMPARISONS AND THEN THE C2 RETURN STATEMENT

0 6 mid = 3
C1

16 26 30 66 76 77 93
Lo Hi

4 6 6 $\frac{6+4}{2}$ = mid = 5
C1

16 26 30 66 76 77 93
Lo, Hi

+7

mid = 6
BREAK
C2

+10

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:

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