Northern Michigan University (Marquette Co, MI)

CS422-01-25W: Algorithms (Andrew A. Poe) Name: ______________________
Practice Midterm Examination (Exam 1) Monday 24 February 2025 9:00 A.M. EST

Time: 50 minutes

1. Given the following classes:

```
public class LL {
 public Node head;
}
```

```
public class Node {
 public String s;
 public Node next;
}
```

Write the code for the method `public String SecondLast (){...}` in the LL class. This method returns the second-to-the-last string in alphabetical order. For example, if the linked list were: I-->ATE-->A-->SANDWICH , this method would return "I". Be sure you handle all special cases in a reasonable way. In particular, if there are two nodes with identical strings and that is the last string in alphabetical order, then that is the string you return since there is a “tie” for last.
You may assume that you have reasonable constructors and sets and gets. You are allowed to use helper methods, but you must write them if you do. Don't use loops; use recursion only. And make no modifications to the linked list or to any of its nodes.

```
public String SecondLast () {

  int sl = 0;
  if (head != null && head.getnext() != null) {
   LinkedListNode l = head.LastExcept (null);
   LinkedListNode sln = head.LastExcept (l);
   sl = sln.getvalue();
  }
  return sl;
 }

public LinkedListNode LastExcept (LinkedListNode exc) {

  LinkedListNode l = null;
  if (next != null) l = next.LastExcept (exc);
  if (exc != this && (l == null || value.compareTo (l.gets())>0))
   l = this;
  return l;
 }
```

2. Write the code for the following method: `public int DelDuplicates (String[] A) {...}`. The parameter A is an array of string that is ALREADY KNOWN TO BE SORTED. However, it may have duplicates. You are to remove the duplicates with in A. For example, if A is a six-element array: {APPLE , APPLE , BOB , CAT , CAT , CAT} . You are to reconfigure the array so that it no longer has duplicates: {APPLE , BOB, CAT , ?? , ?? , ??} (Since the array is of size 6 and there are only three distinct entries, I don't care what the final three entries are.) You are to return the number of distinct entries, in this case, 3. You can use

helper methods but write them if you do. For this one, you can use loops or recursion or both; however, you can't use a second array. Move everything around in A.

```
public int DelDuplicates (String[] A) {

 int np=0;
 if (A.length > 0) {
  for (int i=1; i < A.length; i++)
   if (A[i].compareTo (A[np]) > 0)
    A[np++] = A[i];
  np++;
 }
 return np;
}
```

3. I have a heap that looks like this: G F C D E A B. Pop the heap. What element gets popped off? What does the heap look like afterward? Show all steps.

G F C D E A B Popping the heap, means popping the top of the heap (G) and replacing it with the last element in the heap.

B F C D E A G

B is at position 0; its children are positions 1 and 2, F and C. The largest of the three is F, so we swap B and F.

F B C D E A G

B is at position 1; its children are at positions 3 and 4, D and E. The largest of the three is E, so E and B swap.

F E C D B A G

B is at position 4; its children are at 8 and 9, out of bounds. So B is fine where it is, and the heap is F E C D B A. The G is no longer in the heap because we popped it off, but its still in the array in case we need it.

4. Imagine I have an array that is already sorted (it is, but I don't know that it is). Consider the sorts we have discussed in class: Merge, Shell, Tree, Quick, Heap, Insertion. Which of these sorts runs considerably quicker if the array is already sorted. Which run marginally or “sort of” quicker? Which run about the same? Are there any that run SLOWER? Explain your answer; don't just make a table!!

Insertion runs much quicker when the array is already sorted. Insertion Sort is way too slow to use in general (O(n^2)) but if the array is sorted or very nearly sorted it runs in O(n) time. This is why Insertion Sort is used by Shell Sort. Shell Sort uses Insertion Sort repeatedly to sort a whole bunch of nearly sorted arrays.

Merge is pretty much the same whether or not the array is sorted. The same lists get split. The same lists get merged.

Quick is pretty much the same whether or not the array is sorted. The random selection of the pivot pretty much equalizes all initial permutations. You can be lucky or unlucky with the choice of the pivot, but the initial ordering doesn't matter.

Tree is pretty much the same whether or not the array is sorted. Every item still has to bubble up the same tree no matter what the initial order is.

Shell might actually run a little bit faster if the list is completely sorted. Shell will make the same number of passes no matter what, but if the array is already sorted there won't be any swaps.

Heap might actually run a little bit slower if the array is completely sorted. Since a Heap is similar to the array in reverse order, making the initial heap might be a little bit slower if the array is sorted to begin with.