CS427 Fall Assignment #4
Student Name: _____

Rules:

- Each pseudocode must specify its input and output.
- Necessary comments should be included.
- When analyzing complexity, specific reasons must be provided.
- For proof questions, a specific derivation process is required. Please refer to the example in the slide.

1. Describe the QuickSort algorithm and provide pseudocode and time/space complexity analysis. (20pts)
(Please compare the time complexity from best-case, worst-case and average-case)

/*
In Quicksort:

- You designate a pivot value (typically middle or last value for best results)
- You partition the data by moving all data that's smaller than the pivot to the left
- The pivot is then placed in its proper position
- This is repeated on the left and right subarrays until the subarray size is less than 1

*/

```
//Input: unsorted array, the end and start of the subarray
//Output: the sorted array in memory
quicksort(int[] arr, int low, int high) {
        if(low < high) {
                int pivot = arr[high]  // pivot is the last element
                int i = low – 1; // Index of smaller element

        // Partition
        for (int j = low; j < high; j++) {
                if(arr[j] < pivot) {
                        i++;
                        int temp = arr[i];
                        arr[i] = arr[j];
                        arr[j] = temp;
                }
        }

        // Reassign pivot in correct position
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        p = i + 1; // pivot index after partitioning

        // Recursively sort elements before and after pivot
        quicksort(arr, low, p – 1);
        quicksort(arr, p + 1, high);

        }

}

/* Time Complexity
        Best: O(n log n) – Partition splits are balanced
        Average: O(n log n) – Randomized Data
        Worst: O(n^2) – Already sorted or equal elements
Space:
        O(n) – Requires O(n) stack frames in the worst case
*/
```

2. Merge Two Sorted Lists (20pts).

You are given the heads of two **sorted linked lists** list1 and list2.

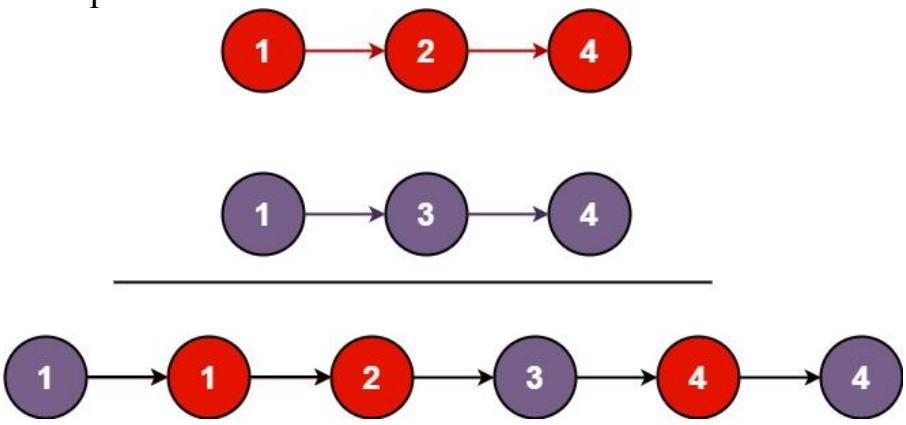
Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Please provide your algorithm, necessary comments and time/space analysis.

(Pseudocode will be fine).

Example 1:



Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:
Input: list1 = [], list2 = []
Output: []

Example 3:
Input: list1 = [], list2 = [0]
Output: [0]

Constraints:
The number of nodes in both lists is in the range [0, 50].
-100 <= Node.val <= 100

Both list1 and list2 are sorted in non-decreasing order.

```
// Input: Two sorted linked list heads
// Output: A single merged sorted linked list

mergeLists(ListNode list1, ListNode list2) {
        // Safe fake head to attach nodes, will be removed after linking
        ListNode fakeHead = new ListNode(-1);
        // Each node of the merged list will be attached at the tail
        ListNode tail = fakeHead;

        // Compare which list has the smallest node value
        while(list1 != null && list2 != null) {
                if(list1.val <= list2.val) {
                        tail.next = list1;
                        list1 = list1.next;
                } else {
                        tail.next = list2;
                        list2 = list2.next;
                }
                tail = tail.next;
        }

        // Attach the rest of the nodes if one list is larger than the other
        tail.next = (list1 != null) ? list1 : list2;

        // Remove the reference to the fake head and return the rest of the merged
list
        return fakeHead.next;

}

// Time O(n + m) – All nodes in both unique lists are examined once
// Space O(1) – Node pointers only change their reference so space remains
constant
```

3 Find Peak Element (20pts)
A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array “nums”, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

Please provide your algorithm, necessary comments and time/space analysis.

(Pseudocode will be fine).

Example 1:

Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Constraints:
1 <= nums.length <= 1000, -231 <= nums[i] <= 231 - 1
nums[i] != nums[i + 1] for all valid i.

\

```
// Input: An array of integers
// Output: The peak element based on its neighbors
findPeakElement(int[] nums) {

        int left = 0;
        int right = nums.length-1;

        while(left < right) {
                int mid = left + (right - left) / 2;

                if(nums[mid] < nums[mid + 1]) {
                        // Peak is still to the right
                        left = mid + 1;
                } else {
                        // Peak is to the left
                        right = mid;
                }
        }

// left = right points to peak index
        return left;

}

// Time Complexity: O(log n) – Each iteration will halve the search space
// Space Complexity:; O(1) – No extra data structures or stack frames used
```