Grading Guidelines for Part-1 of Lab Test 2

Problem Set 1

Grade this part strictly, do not be too lenient. For 1 mark subparts, if they are not close, give 0.

If they do not submit the Intermediate file and were not excused by us: deduct -1 for problem set 1, 0.5 for problem set 2)

- Output: 2 marks
    - No marks in output for printing the intervals
    - Exact format not important.
    - Enter inputs exactly in this order, we want to test that they do not delete any interval ahead of time (which can result in removing the only overlap of another interval later)
    - Test case 1: [1.5, 4], [2.5, 3.5], [4, 6.5], [7.5, 9], [0.5, 2], [6, 7]
        - Max overlap interval [1.5, 4], no. of overlap = 3 (0.5 marks, binary, they must get both correct)
        - Set after deletion [7.5, 9] (0.5 marks, binary)
    - Test case 2: [1, 2], [2, 3], [4, 5], [6, 7], [1.5, 2.5], [1, 1.25]
        - Max overlap interval: [1, 2], no. of overlap = 2 (0.5 marks, binary, they must get both correct)
        - Set after deletion [4, 5], [6, 7] (0.5 marks, binary)
- Code: 8 marks
    - MaxOverlap() function: 3 marks
        - Putting in and returning through proper parameters: 1 mark
            - For parameters, they can use three different pointer parameters or pointer to a single structure that they define additionally or any combination. But they must pass a pointer or give 0 for this part.
        - Finding count of number of overlaps for intervals: 1 mark
        - Keeping track of max: 1 mark
            - Give 0 in this part if they save the counts in an additional array and then do max
        - Deduct 1 if they change the return type from void
    - DeleteInterval() function: 4 marks
        - Going over the intervals to find overlap etc. has no marks in this part, they are already doing it in the MaxOverlap()
        - There are three potential programs that I think you may see for this part. Make sure you understand them before starting grade.
            - The best one is you search for a non-overlapping interval, and when found, swap it with the first (from the beginning) overlapping interval to the left. This is similar to one implementation of quicksort partition (think of the non-overlapping and overlapping

intervals as 0 and 1, and you are trying to get all 0's before 1). Needs no additional array.

- Same as above, but instead of swapping, they just overwrite the first overlapping interval to the left. This is wrong in general, as there may be an interval later which overlaps only with the deleted interval, so now it will show up as non-overlapping. However, you can make it correct if you sort the intervals by the left endpoint first and then consider them in that order.
- The one with one additional array. This is simple, just mark which ones are non-overlapping in one pass, and then delete them in another, shifting array elements to the left as you delete.
- When they delete, it is ok if finally the value of n is finally set to the correct final count and the first n elements of L and R contain the correct points. No need to worry about the extra elements after that. Also, if n =0, it is not required for them to set L, R to null.
- May see others, nothing I can think of now
- Grade based on correctness as per the marks distribution below,

- Marks breakup:
  - Passing a pointer to the INTERVALSET structure (non-pointer will not do, as you may be changing n) - 0.5 marks
  - Correct shifting the intervals around as per above – 3 marks
    - Should change n also properly
    - Make sure they handle the case when two overlapping intervals are at two ends of the L/R arrays
    - Give 0, 1, 2, or 3 only, no 0.5 granularity for this part
  - Handled the case when L and R may have to be set to NULL (no non-overlapping interval) – 0.5 marks
  - If they use any additional array, (i) for one additional array, grade as above, then deduct 2, (ii) for more than one additional array, give 0 in this part.
  - Deduct 1 if they change the return type from void

- main(): 1 mark
  - Proper malloc of the L and R arrays: 1 mark
    - Give 0 if they use static array, or do any other things (some have done malloc inside structure declaration!!)

<u>**Problem Set 2**</u>

Grade Leniently. Use your judgement.

- Output: 1 mark
  - Test case 1: (1, 3), (4, 2), (4, 3), (1, 4), (5, 2)
    - Number of pairs = 6
    - Area of smallest rectangle = 8

        - Area of largest triangle = 2.0
        - Give 0.5 if any two are correct
- Code: 5 marks
    - CountNonHV() function: 2 marks
    - FindArea() function: 1 mark
    - MaxArea function(): 1 mark
        - Return type of this was changed to double during the test, so ok if they take double., or had just kept as int
    - main(): 1 mark
        - reading and printing the points – 1 mark
    - For each, give most of the marks as long as they are more or less correct. Marks for each part are small, so if they are very close, just some minor mistake, you may think of taking 0.5 over two functions instead of 0.5 in each etc. Use your judgement.