Don’t be satisfied with stories, how things have gone with others. Unfold your own myth.

Rumi

Please pay attentions to the followings when preparing your assignment:

• There is both written and programming components in this assignment.

• You need to submit your solutions for the written parts electronically via Crowdmark. You need to submit your programming questions via UM-learn as you did in Assignment 1. Place all Java files in a single folder that will be available for Assignment 3.

• There are 5 problems with a total mark of 66. All questions are mandatory.

• As always, I encourage you to ask your questions on Piazza. Those students who help others on Piazza will receive bonus marks (by ‘helping’, I mean removing confusions). It is likely that I drop hints on Piazza in response to the questions that are asked publicly. In the pursuit of fairness, I hesitate to drop a hint in response to private questions or emails. So, be active on Piazza!

• This is an individual assignment. You are welcome to discuss questions with your friends (or enemies). But you should write the answers individually, and you should fully understand what you are writing. Please read http://www.cs.umanitoba.ca/~kamalis/winter19/infoCOMP2140.pdf for guidelines on academic integrity.

• Good luck!
**Problem 1  Multiple Parenthesis [5+15 marks]**

A sequence of parenthesis is said to be ‘balanced’ if each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested. For example, ((())) is balanced and (()()) is not. A multiple parenthesis is the same concept except that parenthesis have types and each open symbol has a corresponding close symbol of the same type. For example, ((0)(1)(2)) is balanced while (0(1)2) and (2(0)2) are not balanced. We assume a multiple parenthesis sequence formed by $n$ symbol is presented with two arrays: an array $p$ of integers and a array $b$ of booleans, where $p[i]$ indicate the type of the $i$’th symbol and $b[i]$ indicate whether the $i$’th symbol is an open (false) or a close symbol (true). For example, for ((2)(0)2), $p$ includes [2, 0, 2] and $b$ includes [false, false, true, true].

(a) Explain an efficient strategy for checking whether a given multiple parenthesis sequence is balanced or not. Describe your solution using English words. For an input sequence of length $n$, your algorithm should run in time $O(n)$. Note that you might need to use some of the data structures we learned in the class.

(b) Implement your algorithm from part (a). For that, create a class named `Parenthesis` which have a method `public static boolean checkBalanced(int[] p, boolean[] b)`. The output of the method is `true` if the sequence is balanced and `false` otherwise. In case you use a data structure learned in the class (e.g., a queue or a stack), you have to implement them in your code (that is, you cannot use Java’s libraries).

p.s. A test file will be provided shortly.

**Problem 2  Circular Shifts [15 marks]**

Assume you are given a linked list formed by $n$ nodes such that each node stores an integer. Write an efficient Java code that prints all circular shifts of the odd numbers in the linked list in array of $n$ strings. For example, if the list is 1 → 2 → 15 → 14 → 23, the output will be an array `str` of strings such that `str[0] = “1, 15, 23″`, `str[1] = “15, 23, 1″`, and `str[2] = “23, 1, 15″`. Note that the numbers are separated by ‘,’ and there is no space between them.

You need to create a class named `Shifts` which has the following method: `public static String[] giveShifts(LinkedList<Integer> list)`. Here `list` is a linked list that maintains integers. Refer to slides 13 and 14 (of the module on stacks and linked lists) to see how `Node` and `LinkedList` should be implemented. The output of the method is an array of strings as described. In case you use a data structure learned in the class (e.g., a queue or a stack), you have to implement them in your code (that is, you cannot use Java’s libraries).

p.s. A test file will be provided shortly.

**Problem 3  Detecting the Right Sorting Algorithm [6 × 3 = 18 marks]**

For each of the following questions, a situation is described in which you have to use a sorting algorithm. Among the proposed sorting algorithms, indicate the most suitable algorithm. Note that you can select only one algorithm. For each situation, you have to consider factors such as time complexity, space complexity (e.g., being in-place), and applicability of the sorting algorithms. In all cases, assume $n$ is a very large number. You need to justify your answer in one or two sentences.

(a) A set of $n$ integers in the range $[-100, 100]$ should be sorted.

<table>
<thead>
<tr>
<th>Insertion Sort</th>
<th>Quick Sort</th>
<th>Marge Sort</th>
<th>Counting Sort</th>
</tr>
</thead>
</table>

(b) An array of $n$ real numbers should be sorted. The array was already sorted but a bug in a function developed by a lazy programmer caused only a few of the numbers get out of order. So, the array is almost sorted.

| Insertion Sort | Quick Sort | Marge Sort | Counting Sort |

(c) An array of $n$ real numbers should be sorted. The array is so big that it occupies more than half of the main memory of your laptop. You need to sort the array using your laptop; so you need to be careful about memory usage of your algorithms.

| Quick Sort | Marge Sort | Counting Sort |

(d) We want to sort an array of $n$ integers with values in the range $[0, n \log n]$.

| Marge Sort | Counting Sort |

(e) You want to sort records of $n$ students in University of Manitoba. You are given an array with each index having a pointer to a class Student. In the sorted array, international students appear before Canadian ones. For two international students, the one with the higher GPA appears first. For two Canadian students, the one with lower GPA appears first.

| Insertion Sort | Counting Sort |

(f) You want to sort a stream of integers which arrive one by one in an online manner. As the numbers arrive, you need to sort them without knowing when the whole stream ends.

| Insertion Sort | Quick Sort | Merge Sort |

Marking Scheme: for each part except for (e), 1 mark for the correct answer and 2 marks for the correct justification.

**Problem 4  Quick Sort [8 marks]**

(a) A stable sorting algorithm is one in which the relative order of all identical elements (or keys) is the same in the output as it was in the input. Indicate whether quick sort (as we saw in the class) is stable or not. You need to justify your answer via an example or a proof.

Marking Scheme: This part has 3 marks: 1 mark for mentioning that the algorithm is not stable, 2 for a justification; this justification should include an example. A description without example does not suffice (and gets at most 2 marks).

(b) Consider an implementation of quick-sort in which the pivot is always selected in a way that at least $n/4$ items are smaller than it and at least $2n/3$ items are larger than it. Provide a recursion for the best-case time complexity of the algorithm and solve the recursion to express the best-case time complexity using big-Oh notation.
Problem 5  Lower Bounds [5 marks]

Recall that in a sorted array of \( n \) comparable items, we can use binary search to search for a given item in \( O(\log n) \). Prove that binary search is the optimal searching algorithm in a sorted array. You need to use a decision tree approach to show that no algorithm can search in a sorted array in time less than \( O(\log n) \).