# School of Computer Science

COMP20010: Data Structures and Algorithms I
Semester I

**Assignment 1**
On: 08/11/2018
Due: 16/11/2018

Answer all questions.

**Q1:** **(100points)**

(a) Describe some of the implementation differences between a singly linked list and a doubly linked list. [5]

> ***Solution:*** *Need to mention these 3 points:*
>
> *SLL have one pointer per node, DLL have two (next, prev). The SLL has one pointer node (head), the tail points to null. DLL has two extra nodes, header and trailer.*

(b) Describe some of the differences between a circularly linked list and a singly linked list. [5]

> ***Solution:*** *SLL the tail pointer points to null. CLL the end of the list (tail) points to the head. The CLL has a cursor node, which is usually the end of the list.*

(c) Describe 2 applications where you think a circularly linked list would be useful, and say why you would prefer a circularly linked list to a singly or doubly linked list. [5]

> ***Solution:*** *There are many possible answers, check the examples are reasonable and there is a good reason why CLL is required over a SLL*
>
> *Round robin scheduler : a cpu serving a queue of processes, a natural data structure is a CLL. As each process is served, the cursor is incremented to the next process.*

(d) Write the pseudo-code for a non-recursive method which finds the middle node of a doubly linked list. Your method can *not* use a counter, only link hopping. What is the Big-O complexity of your method? [20]

> ***Solution:*** *start two pointers at the top of the list, with slow and fast pointers:*

```
function findMiddle(LL ll)
    Node slow = ll.head()
    Node fast = ll.head()
    while(fast != null and fast.next() != null) do
        slow = slow.next()          # increment by 1
        fast = fast.next().next()   # increment by 2

    return slow.getData()
```

(e) Write the pseudo-code for an algorithm which concatenates two doubly linked lists, *A* and *B* into a new doubly linked list which contains all the elements of *A* followed by all the elements of *B*. [15]

***Solution:*** *To concatenate two linked lists, you have to make the last node of first list point to first node of the second list.*

*check for getting the tail of the first list, setting the next to head of second list, returning joined list.*

```
function concatentate(LL a, LL b)
    # input 2 linked lists, a and b

    Node a_tail = a.head()
    while(a_tail.next() != null) do
        a_tail = a_tail.next()

    # now a_tail is the tail of a
    a_tail.setNext(b.head())
    return a
```

(f) You have two sorted linked lists *A* and *B*. Write the pseudo-code for an algorithm which takes these two lists and returns a new list: $A \cap B$. Your method should only the standard List ADT methods. [15]

***Solution:*** *intersection: Create an empty result list. Traverse ll_a and look for its each element in ll_b. If the element is present in ll_b, then add the element to result.*

```
function intersection(LL ll_a, LL ll_b)
    LL result = new LL()              # empty list

    Node node_a = ll_a.head()

    while(node_a.next() != null) do     # for each node in a
       Node node_b = ll_b.head()       # look for the element in b
        while(node_b.next() != null) do
            if(node_a.getData() != node_b.getData()) then
         result.addBack(node_a.getData())   # if we find it in b then add to result
                break

        node_b = node_b.next()          # continue iterating on a

    return result
```

(g) You have two sorted linked lists *A* and *B*. Write the pseudo-code for an algorithm which takes these two lists and returns a new list: $A \cup B$. Your method should only the standard List ADT methods. [15]

***Solution:*** *union: Very similar to intersection. Create an empty result list. Traverse ll_a and add all its elements to result. Traverse ll_b and look for it in result. If the element is not present in result, then add the element to result.*

*Key points: the first step of including a full copy of one list into the result. Then searching for each value of ll_b in result.*

```
function union(LL ll_a, LL ll_b)
    LL result = new LL()              # empty list

    Node node_a = ll_a.head()

    # add all elements from a to result
    while(node_a.next() != null) do
        result.addLast(node_a.getData())
        node_a = node_a.next()


    Node node_b = ll_b.head()
    while(node_b.next() != null) do     # for each node in a
       Node node_a = ll_a.head()       # look for the element in b

        found = false
        while(node_a.next() != null) do # look for node_b in ll_a
            if(node_a.getData() == node_b.getData()) then
                found = true            # flip the 'found' switch
                break

        if found == false:
            result.addLast(node_b.getData())

     node_b = node_b.next()       # continue the iteration over ll_b

    return result
```

(h) Write the pseudocode for an algorithm which checks if a singly linked list is a palindrome. The method should return `true` if the list is a palindrome and `false` otherwise. A palindrome linked list is the same forwards as backwards, for example: [20]

```
L = 2, 2, 3, 4, 4, 3, 2, 1 // isPalindrome() == false
L = 1, 2, 3, 4, 4, 3, 2, 1 // isPalindrome() == true
```

***Solution:*** *There are a few different methods that will work. Any is acceptable as long as implemented correctly.*

1. *Reverse make a copy of the list in reverse order. Then iterate through both lists simultaneously and check each node. If all are the same, the list is a palindrome.*

2. *Recursive Use recursion to reverse the list and check the values in reverse order:*

```
function isPalindrome(LL ll)
    return isPalindrome_helper(ll.head(), ll.head())

function isPalindrome_helper(Node left, Node right)
    if right == null then
        return true

    # the first sequence of recursion iteratates right
    # to the end of the list
    #
    # as the recursion is unwound we are effectively moving
    # right from the end of the list to the head
    boolean res = isPalindrome_helper(left, right.next()) \\
                    && left.getData() == right.getData()
    left = left.next()
    return res
```

3. *Use a stack...*

**Q2:** **(100points)**

(a) What is the Big-O complexity of [5]

$$20n^3 + 10n \log n + 5n + 4$$

***Solution:*** $O(n^3)$

(b) The number of operations executed by algorithms $A$ is $8n \log n$ and by $B$ is $2n^2$. Determine $n_0$ such that $A$ is better than $B$ for $n \geq n_0$. [5]

***Solution:*** *We can see that $A$ must be slower than $B$ for very large $n$. With a little rearrangement setting $8n \log n < 2n^2$ we see that $n_0$ is found by the condition $\frac{\log n}{n} < 4$. Can you solve this directly? probably easier to try a few different values (remember its $\log_2$ not $\log_{10}$:*

| $n$ | $\frac{\log n}{n}$ |
|---|---|
| 2 | 0.5 |
| 3 | 0.52832 |
| $\ldots$ | $\ldots$ |
| 15 | 0.260459 |
| 16 | 0.25 |
| 17 | 0.240439 |

*So* $n_0 = 16$.

(c) The number of operations executed by algorithms $A$ is $40n^2$ and $B$ is $2n^3$. Determine $n_0$ such that $A$ is better than $B$ for $n \geq n_0$. [5]

***Solution:*** *Rearrange*: $40n^2 < 2n^3$, $n > 20$. ***So,*** $n_0 = 20$.

(d) What is the sum of all the even numbers from $0$ to $2n$, for any integer $n \geq 1$? [5]

***Solution:***

$$\sum_{0}^{2n} n = \frac{2n(n+1)}{2}$$

(e) Show that if $d(n)$ is $O(f(n))$,then $ad(n)$ is $O(f(n))$, for any constant $a > 0$? [5]

**Solution:**

$$\begin{aligned} d(n) is &\qquad O(f(n)) \\ ad(n) is &\qquad O(af(n)) \end{aligned}$$

and $O(af(n))$ is $aO(f(n))$ since $a$ is a constant.

(f) Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the product $d(n)e(n)$ is $O(f(n)g(n))$. [5]

**Solution:** *The product of upper bounds of functions gives an upper bound for the product of the functions: The product $d(n)e(n)$ has complexity $O(f(n))O(g(n)) = O(f(n)g(n))$.*

(g) What is the big Oh characterisation of the following code: [5]

```java
public static int f() {
        int n = arr.length;
        int total = 0;
        for (int j=0; j < n; j += 2) {
                total += arr[j];
        }
        return total;
}
```

**Solution:** $O(n)$

(h) What is the big Oh characterisation of the following code: [5]

```java
public static int f(int[] first, int [] second) {
  int n = first.length;
  int count = 0;
  for(int i=0; i<n; i++) {
      int total = 0;
      for (int j=0; j < n; j++) {
          for (int k=0; k <= j; k++) {
              total += first[k];
          }
      }
      if (second[i] == total) {
          count++;
      }
  }
  return count;
}
```

***Solution:*** $O(n^3)$

(i) Show that if $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$. [5]

***Solution:** Informally: if $d(n)$ grows as fast as $f(n)$ and $f(n)$ grows as fast as $g(n)$, then $d$ grows as most as fast as $g$. So: $d(n)$ is $O(g(n))$.*

(j) Show that $O(max f(n), g(n)) = O(f(n) + g(n))$. [5]

***Solution:** Informally: the basic rules of complexity analysis: the sum grows as fast as the fastest growing term.*

(k) Show that if $p(n)$ is a polynomial in $n$, then $\log p(n)$ is $O(\log n)$. [5]

***Solution:** If $p(n)$ is a polynomial of degree $d$: $p(n,d) = a_0 + a_1 n + \ldots + a_d n^d$, then the largest term in $\log(p(n))$ will be $\log n^d$. But $O(\log n^d) = O(d \log n) = dO(\log n) = O(\log n)$, since $d$ is a constant.*

(l) Show that $2^{n+1}$ is $O(2^n)$. [5]

***Solution:** $a^{x+y} = a^x a^y$ so $2^{n+1} = 2^n 2^1$. The big-O complexity of $2^n 2^1$ is $O(c2^n) = O(2^n)$.*

(m) Algorithm $A$ executes an $O(\log n)$-time computation for each entry of an array storing $n$ elements. What is its worst-case running time of $A$? [5]

***Solution:** The worst case running time is: $O(n \log n)$.*

(n) Given an $n$-element array $X$, Algorithm $B$ chooses $\log n$ elements in $X$ at random and executes an $O(n)$-time calculation for each. What is the worst-case running time of Algorithm $B$? [5]

***Solution:** The worst case is that the random choice chooses $n$ elements, so the worst case running time is: $O(n \log n)$.*

(o) Alice and Bob are arguing about their algorithms. Alice claims her $O(n \log n)$-time method is always faster than Bob's $O(n^2)$-time method. To settle the issue, they perform a set of experiments. To Alice's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$ is the $O(n \log n)$-time one better. Explain how this is possible. [10]

***Solution:** I am looking for some key points in the argument: Alice's algorithm is faster for large $n$, but we can adjust the constants to make is slower for $n < n_0$.*

*The $O(n \log n)$ algorithm is only guaranteed to be faster for infinitely large $n$. The actual run time complexity is $c_{alice}T(n \log n)$ and $c_{bob}T(n^2)$, and the ratio of the constants $c_{bob}/c_{alice}$ can be arranged to make sure that $n_0 < 100$.*

(p) An array $A$ contains $n-1$ unique integers in the range $[0, n-1]$, that is, there is one number from this range that is not in $A$. Design an $O(n)$-time algorithm for finding that number. You are only allowed to use $O(1)$ additional space besides the array $A$ itself. [10]

***Solution:*** *There are lots of ways to do this. The wrong way is to sort the array, which would be at least $O(n \log n)$ or $O(n^2)$. Nested for loops is also wrong, if they lead to $O(n^2)$ complexity. One solution is to sum the values in the array. We know one value is missing and we know what we expect, so the missing number is the difference between the two:*

```
actual_sum = sum(A)                       # O(n)
expected_sum = (n+1) n / 2                # const
missing_value = expected_sum - actual_sum # const
```

*this method is total $O(n)$.*

(q) The following pseudo-code describes a well-known algorithm. Describe what it does? Annotate the algorithm and determine its asymptotic running time complexity: [10]

```
function FOO($A, n$)
    Input: array $A$ of size $n$
    for $i =$ to $n - 1$ do
        $min \leftarrow i$
        for $j = i + 1$ to $n$ do
            if $list[j] < list[min]$ then
                $min \leftarrow j$
            end if
        end for
        if $min \neq i$ then
            swap($list[min]$, $list[i]$)
        end if
    end for
end function
```

***Solution:*** *This is selection sort and is $O(n^2)$ complexity.*