# COMP20010
# Data Structures and Algorithms Assignment 1

Q1.A)
When Implementing a singly linked list your Node class must have just a next Node variable But in your Doubly Linked list your Node class will have  next and previous node variables. The doubly linked list Class should have an insertBetween routine. The singly linked list should have only one Node variable tabled head whereas the Doubly Linked List Class will have two Node one head and one tail.

Q1.B)
A singly linked list only has one reference head node.
A doubly linked has two reference nodes a head and a tail.
A doubly Linked List has a method to insert nodes between each other. The singly linked list nodes only have one node variable.

Q1.C)
-Keeping track of turns for a multiplayer board game,
-Applications on a pc. When you tab between them the list is circular.

Q1.D)
```
function middleNode(x, y)
        Input: x head node of list
               y tail node of list
        Output: middle node of list
        if (x or y = null)
                return null
        while (true)
                if(x = y)
                        break loop
                x := x.next
                if (x = y)
                        break loop
                y := y.previous
        return x
end function
```

The complexity of this algorithm is O(n)

Q1.E)
```
function concatLists (A, B)
        input: a - a doubly linked list
               b - a doubly linked list
        output: A concatenated with B in a new Doubly linked list
        concated := new doublyLinkedList
        ahop := A.first
        bhop := B.first
        while (ahop != null)
                concated.addlast(ahop.data)
        while (bhop != null)
                concated.addlast(bhop.data)
        return concated
end function
```

Q1.F)

```
 function aintersectb(A, B)
	input: A - sorted linked list
	       B - sorted linked list
	output the intersection list of A, B
	intersect := new linkedList
	ahop = A.first
	bhop = B.first
	while (ahop != null)
		while (bhop != null)
			if (bhop.data = ahop.data)
				intersect.addlast(bhop.data)
				break loop
			shop := bhop.next
		ahop := ahop.next
	return intersect
end function
```

Q1.G)

```
function aunionb(A, B)
	input: A - a sorted linked list
	       B - a sorted linked list
	output: the union of A, B
	union := new linkedList
	ahop := A.first
	bhop := B.first
	while (ahop != null)
		union.addlast(ahop.data)
	while(bhop != null)
		isIn := new boolean (false)
		ahop :=A.first
		while(ahop != null)
			if (ahop.data = bhop.data)
				isIn := true
				break loop
			ahop := ahop.next
		if(!isIn)
			union.addlast(bhop.data)
		bhop := bhop.next
	return union
end function
```

Q1.H)

```
function checkListisPalindrome(A)
	input: A - a singly linked list
	output: Boolean value
	B := A.copy.reverse // copies and reverses A and stores it in B
	ahop := A.first
	bhop := B.first
	while(ahop != null)
		if (ahop.data != bhop.data)
			return false
		ahop := ahop.next
		bhop := bhop.next
	return true
end function
```

Q2.A)
$20n^3 + 10nlogn + 5n + 4$
= {Drop Constant}
$20n^3 + 10nlogn + 5n$
= {Drop co-efficients}
$n^3 + nlogn + n$
= {Take highest degree term}
$n^3$

$20n^3 + 10nlogn + 5n + 4$ is $O(n^3)$

Q2.B)
Algorithm

$A : 8nlogn$

$B : 2n^2$

$8nlogn = 2n^2$
= {divide by 2}
$4nlogn = n^2$
= {divide by n}
$4logn = n$
= {divide by 4}
$logn = \frac{n}{4}$
= {get rid of log}
$2^{\frac{n}{4}} = n$

Now by trial and error we will try get $2^{\frac{n}{4}} = n$
Using an educated guess and that $2^{\frac{n}{4}} = n$ wont be true until at least n = 12
We try values from 12 - 20.
We find that at n = 16 $2^{\frac{n}{4}} = n$
We conclude that Algorithm A is better than B for $n \geq 17$

Q2.C)
Algorithm

$A : 40n^2$

$B : 2n^3$

$40n^2 = 2n^3$
= {divide by n squared }
$40 = 2n$
= {divide by 2}
$n = 20$

A = B at n = 20
So Algorithm A is better than Algorithm B for $n \geq 21$

Q2.D)
Sum of all integers from 0 to n $= \frac{n(n+1)}{2}$

This is 1 + 2 + … + n

But we want 2 + 4 + … + 2n
= 2(1 + 2 + … + n)
= $2(\frac{n(n+1)}{2})$
= $n(n+1)$

So, the sum of all even integers from 0 to 2n is $n(n+1)$

Q2.E)
d(n) is O(f(n))
Implies that

$$d(n) \leq cf(n) \text{ for all } n \geq n_0$$

When we take some constant a and get ad(n) we have

$$ad(n) \leq af(n)$$

Which implies that

$$ad(n) \leq f(n) \text{ for all } n \geq an_0$$

And by definition then

ad(n) is O(f(n))

Q2.F

d(n) is O(f(n))
e(n) is O(g(n))
Means that

$$d(n) \leq cf(n) \text{ for all } n \geq n_0$$

And

$$e(n) \leq c'g(n) \text{ for all } n \geq n_0'$$

Now for d(n) times e(n)

$$d(n)e(n) \leq cc'f(n)g(n) \text{ for all } n \geq n_0 n_0'$$

Then from definition of big-oh

d(n)e(n) is O(f(n)g(n))

Q2.G)
The for loop , loops n times but increments in 2 so it actually loops $\frac{n}{2}$ times

O($\frac{n}{2}$) is O(n)

Since $\frac{n}{2}$
= $\frac{1}{2}n$

Then we drop the co-effiecient to get O(n)

Q2.H)
First loop is n times.
The second loop is n times
The third loop is $n^2$ times

We can simplify the second and third loops to $n^2$, then since the outer loop is n times
The total is $nn^2$ which is $n^3$ times
So the algorithm is O($n^3$)

Q2.I)
d(n) is O(f(n))
Implies
$d(n) \leq cf(n)$ for all $n \geq n_0$
And
f(n) is O(g(n))
Implies
$f(n) \leq c'g(n)$ for all $n \geq n_0'$
We combine both to get
$d(n) \leq cf(n) \leq c'g(n)$
From this we can say
$d(n) \leq c'g(n)$ for all $n \geq n_0'$
By definition
d(n) is O(g(n))

Q2.J)

If O(Max( f(n),g(n) )) is O(f(n)+g(n))
Then
$Max(f(n), g(n)) \leq f(n) + g(n)$ for all $n \geq n_0$
Since out functions are positive their sum will always be greater than the max of the two.
($a \leq a + b$ and $b \leq a + b$ for all $a, b \geq 0$)

Then by definition
$Max(f(n), g(n))$ is $o(f(n) + g(n))$

Q2.K)

If p(n) is a polynomial in n then we will write
$p(n) = n^m$
Then
$log(p(n))$
= {sub in p(n)}
$log(n^m)$
= $\{log(n^k) = klog(n)\}$
$mlog(n)$

Then we drop the co-effiecient m to get the run time.
$log(p(n)) = O(log(n))$

Q2.L)
$2^{n+1}$
= {rule of indices}
$2^1 \cdot 2^n$
= {drop co-efficient}
$2^n$

Q2.M)
Well the algorithm is going to preform the O(log n) computation for each element of an array sized n.
Worst case running time will be: n log n

Q2.N)
The worst case run time is: n log n
Explanation:
The algorithm is choosing log n elements but each time it does so it performs a n time calcution. So the time complexity is log n times n which is n log n.

Q2.O)
This is possible because even though alices algorithm is O(n log n) it can still contain many other lower order calculations which we ignore when calculating the overall time complexity.

Q2.P)
```
function getMissingNumber(A, n)
        input: A - an array of integers
               n - the size of the array
        output: the integer missing from the array
        total := (n(n + 1))/2
        for j in range (n)
                total -= A[j]
        return total
end function
```

Q2.Q)
The algorithm is selection sort.

It divides the array into two parts that consist of the sorted items on the left and unsorted items on the right. It finds the smallest element in the unsorted part of the array and swaps it with the left most element of the unsorted part then expands the sorted part to contain that new element. It keeps doing this until the list is fully sorted.

The time complexity of insertion sort is $O(n^2)$