Assignment 2

1 – a) Recursive Algorithm that reverse the elements in the singly linked list

```
Algorithm reverse (head)
    If head  = null
    return
    If head.next = null
    return head
    restElems ← head.next
    reverse (restElems)
    restElems.next ← head
    head.next ← null
return restElems
 end
```

b) Recurrence equation of the running time

$$T(n) = \begin{cases} 2 \text{ if } n = \text{length of list} \\ 3 \text{ if } n = 1 \\ 6 + T(n-1) \text{ if } n > 1 \end{cases}$$

c) Determine big oh by determining the closed form

$T(n) = 6 + T(n-1)$

$T(1) = 3$

$T(2) = 6 + T(1) = 6 + 3$

$T(3) = 6 + T(2) = 6 + 6 + 3$

T(4) = 6 + T(3) = 6 + 6 + 6 + 3

T(n) = 6 (n-1) + 3 = 6 n – 3

T(n) = 6n -3 if there is at least one element in the list. So , it is linear, O(n).

2 – a) Iterative Algorithm that reverse the elements in the singly linked list

```
Algorithm Reverse (head)
prevNode ← null
nextNode ← null
currentNode ← head
while (currentNode != null) do
    nextNode ← current.next
   current.next ← prevNode
   prevNode ← currentNode
  currentNode ←nextNode
endwhile
head ← prevNode                    set the first element to null
return
end
```

2-b) To convert a given recursive algorithm into an iterative algorithm we can use the stack-based implement due to the nature of nested function calling and returning in recursion. It is the same as stack last in first out. In the nested function called in recursion, the later a function is called, the earliest it returns which is the same as stack last in first out.

3 - Describe in pseudo code an O(n) time algorithm of finding the number

```
Algorithm findNumber(A)
    n ← A.length
    sum ← n * (n+1) / 2
    total ← 0
   for i← 0  to n-2  do
       total ← total + A[i]
  endFor
 return sum – total
end
```

4- a - Recurrence equation of FindMin

$$T(n) = \begin{cases} 4 \text{ if } <= 1 \\ 13 + 2\,T(n/2) \text{ if } n > 1 \end{cases}$$

b- Yes the recurrence equation fit the master theorem a= 2, b = 2 , c = 13 and d = 0

c – Big-Oh of the running time

$a > b^d$, $2 > 2^0$ so $2 > 1$ therefore $O(n^{\log_2 2}) = O(n)$

Big-Oh is O(n)

5 -

a –

Insertion Sort will require 4 comparisons

Merge Sort will require 3 comparisons

b-

Insertion Sort will require 10 comparisons

Merge Sort will require 4 comparisons

C – For insertion Sort, the worst case is $O(n^2)$. If the elements are arranged in inverse sorted order like in 5b, every possible comparison is carried out. The insertion runs as slow as bubble sort. The number of comparison is (n-1) in the first time, (n-2) in the second time, (n-3) in the third time, and so on. The sum is $(n-1)+ (n-2) + (n-3) + \ldots +2+1 = n*(n-1) / 2 = (n^2 - n)/2$. So when n is very large n^2 is the dominant term. Big-Oh is $O(n^2)$ is the running time in the case.