Assignment 2 – Theoretical part

CSC 225
V00806981
Allan Liu

<u>Question 1.</u>

a)

Algorithm Node recursive (Node n)
if (n==null) **then**
  return null
if (n.next==null) **then**
  return n
secondNode ← n.next
remainingNode ← recursive (secondNode)
secondNode.next ← n
n.next ← null
return remainingNode

b)

Base case:

If statement: 1 comparison, 1 return when n==null.
If statement: 2 comparison, 1 return when n.next==null.

Line1: 1 comparison
Line2: 1 comparison
Line3: 1 assignment
Line4: 1 assignment, 1 method call
Line5: 1 assignment
Line6: 1 assignment
Line7: 1 return

$$T(n) = \begin{cases} 2, & n = null \\ 3, & n.next = null \\ & 8 + T(n-1) \end{cases}$$

c)

T(n)=8+T(n-1)
=8+8+T(n-2)
=8+8+8+T(n-3)
=8k+T(n-k)

Question 2.

a)

Algorithm Iterative (Node n)
firstNode ← n
reverseNode ← null
while (firstNode != null)
    secondNode ← firstNode.next
    firstNode.next ← reverseNode
    reverseNode ← firstNode
    firstNode ← secondNode
return reverseNode

b)

In general, to convert a recursive algorithm to an iterative one, you implement at loop that will iterate until the base case of the recursion is reached. Instead of sending arguments to the recursive call, send them to the start of the loop instead and progress towards the base case.

Question 3.

Algorithm findNum(A,n)
A: array [0…n-1]
sum ← 0
for i ← 0 to n-1
    sum ← sum + A[i]
end
total ← (n-1) * (n-1)/2
num ← total – sum
return num

Question 4.

a)

Let c_1 and c_2 be constants.

Base case:

If statement: 1 subtraction, 1 comparison, 1 array indexing, 1 return = 4 or a constant c_1.

Else statement: 3 assignments, 1 addition, 1 division, 2 recursion call.
Return 1.

$$T(n) = \begin{cases} c_1, & n = 1 \\ 2T\left(\frac{n}{2}\right) + c_2 n, & n > 1 \end{cases}$$

b)

Yes, this recurrence equation fits the master theorem. Value a=2, b=2, c=8, d=1.

c)

O(nlogn)

Running time = time for all divide operations + FindMin operations. FindMin total time=$c_1 nlogn$, while finding the midpoint takes $c_2 n$ times. So $c_1 nlogn + c_2 n$ in big O notation, is O(nlogn).

Question 5.

a)

Insertion sort: 4 comparisons

Merge sort: 7 comparisons

b)

Insertion sort: 14 comparisons

Merge sort: 5 comparisons

c)

Suppose that every call of insert, the value that is being inserted is always less than the element to its left, that is the worst case. Inserting the first time k=1, then k=2, then k=3 to …k=(n-1). So the time it takes is *c (1+2+3+…+(n-1)).* Using arithmetic series formula: $S_n = n(\frac{a_1+a_n}{2})$

$$=c[(n-1)\left[\frac{(1+(n-1))}{2}\right]]$$

$$=c[(n-1)\left(\frac{n}{2}\right)]$$

$$=c[\frac{n^2}{2}-\frac{n}{2}]$$

Using big O notation, we don't count the constant and lower-order terms, so we get ***O(n^2).***