CURTIN UNIVERSITY

School of Electrical Engineering, Computing, and Mathematical Sciences

Discipline of Computing

Test 1 – Semester 1 2019

SUBJECT: Design and Analysis of Algorithm COMP3001

TIME ALLOWED:

55 minutes test. The supervisor will indicate when answering may commence.

AIDS ALLOWED:

To be supplied by the Candidate:	Nil
To be supplied by the University:	Nil

Calculators are NOT allowed.

GENERAL INSTRUCTIONS:

This paper consists of Two (2) questions with a total of 50 marks.

ATTEMPT ALL QUESTIONS

Name: ____

Student No: ____

Tutorial Time/Tutor: ___________________________________

QUESTION ONE (24 marks)

a) **(Total: 6 marks).** Consider $T(n) = n^3 - 2n^2 + 3n - 4$. Prove that $T(n) = \Theta(n^3)$.

Answer:

b) **(6 marks).** Prove by induction that for $n \geq 1$,

$\sum_{i=1}^{n} i(i+1) = n(n+1)(n+2)/3$

Note: $\sum_{i=1}^{n} i(i+1) = 1\cdot2 + 2\cdot3 + 3\cdot4 + \ldots + n(n+1)$

Answer:

c) **(Total: 6 marks).** Consider the following recursive function.

```
AAA ($n$)
    if $n = 0$ then
        return 0
    else
        return $2n$ + AAA($n - 1$) – 1
```

(i) **(4 marks).** What is the output of the algorithm for $n = 4$? Show your detailed calculation.

(ii) **(2 marks).** Give the recurrence function of the algorithm in terms of n. Explain your answer.

Answer:

(i) $n = 4$

(ii)

d) **(Total: 6 marks).** Consider recurrence function $T(n) = T(n - 1) + 1$.

(i) **(2 marks).** Use a recurrence tree to guess the asymptotic upper bound time complexity of the algorithm.

(ii) **(4 marks).** Use induction to show that your guess is correct.

Answer:

(i) Recurrence tree

(ii) Induction proof.

END OF QUESTION ONE

QUESTION TWO (Total: 26 marks).

a) **(Total: 9 marks).** Consider the following recursive **MERGESORT** algorithm and **MERGE** algorithm to merge two sorted sub-arrays.

MERGESORT(*A*, *l*, *r*)
Input : an array *A* in the range 1 to *n*.
Output: Sorted array *A*.

```
if $l < r$
    then $q \leftarrow \lfloor (l+r)/2 \rfloor$
        MERGESORT($A$, $l$, $q$)
        MERGESORT($A$, $q$+1, $r$)
        MERGE ($A$, $l$, $q$, $r$)
```

MERGE(*A*, *l*, *m*, *r*)
Inputs: Two sorted sub-arrays *A*(*l*, *m*) and *A*(*m*+1, *r*)
Output: Merged and sorted array *A*(*l*, *r*)

```
$i$ = 1
$j$ = $m$+1
$k$ = 1
while ($i \le m$) and ($j \le r$) do      // check if not at end of each sub-array
     if $A[i] \le A[j]$ then              // check for smaller element
         TEMP[$k$++] = A[$i$++]
     else                                  // copy smaller element
         TEMP[$k$++] = A[$j$++]    // into temp array
while ($i \le m$) do
     TEMP[$k$++] = $A$[$i$++]       // copy all other elements
while ($j \le r$) do            // to temp array
     TEMP[$k$++] = $A$[$j$++]
```

Suppose you are asked to modify the MERGESORT algorithm. In the modified algorithm, you divide the number of elements into three parts, instead of two in the original MERGESORT. Then, you merge the three sorted parts.

(i) **(4 marks).** Write the pseudocode of the modified MERGESORT.
Hint. How to merge three sorted parts?

(ii) **(2 marks).** Give the recurrence function for the time complexity of the modified MERGESORT. Explain your answer.

(iii) **(3 marks).** Does the modified MERGESORT has better time complexity than the original MERGESORT? Justify your answer by computing the time complexity of the modified MERGESORT.

Answer:

(i) Modified MERGESORT

(ii) Recurrence

(iii)

b) **(Total: 8 marks).** Consider the following PARTITION algorithm.

```
PARTITION(A, l, r)
Input: Array A(l .. r)
Output: A and m such that A[i] ≤ A[m] for all i ≤ m and A[j] > A[m] for all j > m
  x = A[r]
  i = l – 1
  for j = l to r – 1 do
     if A[j] ≤ x then
       i = i +1
           exchange A[i] ↔ A[j]

  exchange A[i] ↔ A[r]
  return i
```

(i) **(4 marks).** Illustrate the operation of the algorithm on an array *A* = <13, 19, 9, 5, 12, 21, 7, 4, 11, 2, 6, 8>.

(ii) **(2 marks).** What is the best-case lower bound time complexity of the algorithm? Why?

You are NOT asked to formally prove your answer. A short argument is sufficient.

(iii) **(2 marks).** What is the worst-case upper bound time complexity of the algorithm? Why?

You are NOT asked to formally prove your answer. A short argument is sufficient.

<u>**Answer:**</u>

(i)

(ii)

(iii)

c) **(Total: 9 marks).** Let *A* and *B* be two $n \times n$ matrices, where *n* is a power of 2, and $C = A \times B$. Using the divide and conquer method, we divide *A* and *B*, and their product *C* each into four $n/2 \times n/2$ matrices, and thus $C = A \times B$ becomes

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

(i) **(3 marks).** Using the conventional matrix multiplication method, we have:

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$ $C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$
$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$ $C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$

Write the recurrence function for the time complexity of the conventional matrix multiplication method. Explain your answer.

(ii) **(3 marks).** The following is the Strassen's algorithm. The recurrence of the time complexity of the Strassen's algorithm is $T(n) = 7\ T(n/2) + 18(n/2)^2$. Show that Strassen's algorithm has a time complexity of $\Theta(n^{2.807})$

Strassen (*A*, *B*)

```
1   $n = rows[A]$
2   Let $C$ be a new $n \times n$ matrix
3   **if** $n = 1$
4       $c_{11} = a_{11} \times b_{11}$
5   **else**
6   $P_1$ = **Strassen** $(A_{11}, B_{12} - B_{22})$
7   $P_2$ = **Strassen** $(A_{11} + A_{12}, B_{22})$
8   $P_3$ = **Strassen** $(A_{21} + A_{22}, B_{11})$
9   $P_4$ = **Strassen** $(A_{22}, B_{21} - B_{11})$
10  $P_5$ = **Strassen** $(A_{11} + A_{22}, B_{11} + B_{22})$
11  $P_6$ = **Strassen** $(A_{12} - A_{22}, B_{21} + B_{22})$
12  $P_7$ = **Strassen** $(A_{11} - A_{21}, B_{11} + B_{21})$
13  $C_{11} = P_5 + P_4 - P_2 + P_6$
14  $C_{12} = P_1 + P_2$
15  $C_{21} = P_3 + P_4$
16  $C_{22} = P_5 + P_1 - P_3 - P_7$
17  **return** $C$
```

$S_1 = B_{12} - B_{22}$ $S_2 = A_{11} + A_{12}$ $S_3 = A_{21} + A_{22}$ $S_4 = B_{21} - B_{11}$ $S_5 = A_{11} + A_{22}$
$S_6 = B_{11} + B_{22}$ $S_7 = A_{12} - A_{22}$ $S_8 = B_{21} + B_{22}$ $S_9 = A_{11} - A_{21}$ $S_{10} = B_{11} + B_{12}$

$P_1 = A_{11} \times S_1$ $P_2 = S_2 \times B_{22}$ $P_3 = S_3 \times B_{11}$ $P_4 = A_{22} \times S_4$
$P_5 = S_5 \times S_6$ $P_6 = S_7 \times S_8$ $P_7 = S_9 \times S_{10}$

$C_{11} = P_5 + P_4 - P_2 + P_6$ $C_{12} = P_1 + P_2$ $C_{21} = P_3 + P_4$ $C_{22} = P_5 + P_1 - P_3 - P_7$

(iii) **(3 marks).** According to Strassen, $C_{11} = P_5 + P_4 - P_2 + P_6$. Prove its correctness.

Answer:

(i)

(ii)

(iii)

END OF QUESTION TWO

Attachment

Assume the following:

*lg*3 ≈ 1.5, *lg*5 ≈ 2.3, *lg*6 ≈ 2.5, *lg*7 ≈ 2.8, *lg*9 ≈ 3.1, *lg*10 ≈ 3.3.

Master Theorem:

if $T(n) = aT(n/b) + f(n)$ then

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a-\varepsilon}\right) \rightarrow f(n) < n^{\log_b a} \\ \Theta\left(n^{\log_b a}\lg n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \rightarrow f(n) = n^{\log_b a} \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right) \rightarrow f(n) > n^{\log_b a} \\ & \text{if } af(n/b) \le cf(n) \text{ for } c < 1 \text{ and large } n \end{cases}$$

Quicksort(*A*,*l*,*r*)
Input : Unsorted Array (*A*,*l*,*r*); **Output :** Sorted subarray *A*(0..*r*)

```
if $l < r$
  then $q \leftarrow$ PARTITION($A,l,r$)
      QUICKSORT($A,l,q$-1)
      QUICKSORT($A,q$+1,$r$)
```

PARTITION(*A*, *l*, *r*)
Input: Array A(*l* .. *r*)
Output: *A* and *m* such that *A*[*i*] ≤ *A*[*m*] for all *i* ≤ *m* and *A*[*j*] > *A*[m] for all *j* > m

```
  $x = A[r]$
  $i = l - 1$
  for $j = l$ to $r$ - 1 do
     if $A[j] \le x$ then
           $i = i + 1$
           exchange $A[i] \leftrightarrow A[j]$

  exchange $A[i] \leftrightarrow A[r]$
  return $i$
```

SELECTION_SORT (*A*[1, …, *n*])

Input: unsorted array *A* **Output:** sorted array *A*

```
1. for i ← 1 to n-1
2.     small ← i
3.     for j ← i+1 to n // Set small as the pointer to the  smallest element in A[i+1..n]
4.         if A[j] < A[small] then
5.             small ← j
6.     temp ← A[small]    // Swap A[i] and smallest
7.     A[small] ← A[i]
8.     A[i] ← temp
```

INSERTION-SORT (*A*)

```
for j=2 to length(A) do
    key = A[j]
    // insert A[j] into the sorted sequence A[1 … j-1]
    i = j-1
    while i>0 and A[i] > key do
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key
```

Counting-Sort (*A*, *B*, *k*)

```
1   let C [0 .. k] be a new array     // note that the index of array C starts from 0
2   for i = 0 to k
3       C [i] = 0
4   for j = 1 to A.length
5       C [ A [j] ] = C [ A [j] ] + 1
6   // C [i] now contains the number of elements equal to i
7   for i = 1 to k
8       C [i ] = C [i] + C [i - 1]
9   // C [i] now contains the number of elements less than or equal to i
10  for j = A.length downto 1
11      B [ C [ A [j] ]] = A [j]
12      C [ A [j] ] = C [ A [j] ] - 1
```

END OF PAPER