CURTIN UNIVERSITY

School of Electrical Engineering, Computing, and Mathematical Sciences

Computing Discipline

Test 1 – Semester 1 2018

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 (26 marks)

a) **(Total: 4 marks).** Let $f(n) = n^3 + n^2\lg n + 10$.

(i) **(2 marks).** Is $f(n)= O(n^3)$? Note that you must show the values of c and n_0 to prove its correctness.

(ii) **(2 marks).** Is $f(n)= \Omega(n^3)$? Note that you must show the values of c and n_0 to prove its correctness.

Answer:

(i)

(ii)

b) **(4 marks).** Use the recursion tree to guess the time complexity of the following recurrence.

$$T(n) = T(n-1) + \lg n$$

Answer:

c) **(Total: 8 marks).**

(i) **(4 marks).** Can the master theorem be applied to the following recurrence?

$$T(n) = T(n/2) + n^2$$

If it can, compute the tight asymptotic bound of the recurrence. Otherwise, show why the master theorem cannot be used to produce its solution.
<u>Note:</u> you must use the polynomial and regularity checks when necessary.

(ii) **(4 marks).** For recurrence $T(n) = 2T(n/3) + n^2$, use induction to prove or disprove that the solution to the recurrence is $O(n^2)$.

<u>Answer:</u>

(i) Master Theorem

(ii) Induction

d) **(Total: 10 marks).** The following two functions evaluate a polynomial for a given value of $x = c$

$$y = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_1 x + a_0$$

Poly_1 ($c, a_0, a_1, a_2, \dots a_n$)
  $p \leftarrow 1$
  $y \leftarrow a_0$
  for (i = 1 to n)
    $p \leftarrow p * c$
    $y \leftarrow y + a_i * p$

Poly_2 ($c, a_0, a_1, a_2, \dots a_n$)
  $y \leftarrow a_n$
  for (i = 1 to n)
    $y \leftarrow y * c + a_{n-i}$

(i) **(5 marks).** Show the working of each of the two algorithms to evaluate $y = x^2 + 2x + 3$, for $x = 2$. More specifically, for each iteration i, show the values of p and y in **Poly_1**, and the value of y in **Poly_2**.

(ii) **(5 marks).** Calculate the total number of steps to execute each algorithm, express the results in Big Oh, and explain which algorithm is more efficient.

Answer:

(i) **Poly_1:**

Poly_2:

(ii)

Poly_1(c, a_0, a_1, a_2, … a_n)
    $p \leftarrow 1$
    $y \leftarrow a_0$
    for (i = 1 to n)
        $p \leftarrow p * c$
        $y \leftarrow y + a_i * p$

Total steps:

Poly_2 (c, a_0, a_1, a_2, … a_n)
    $y \leftarrow a_n$
    for (i = 1 to n)
        $y \leftarrow y * c + a_{n-i}$

Total steps:

END OF QUESTION ONE

QUESTION TWO (Total: 24 marks).

a) **(Total: 17 marks).** Consider an array A of n integers, and another array B of n integers. The array A contains n consecutive odd numbers in increasing order, i.e., A = <1, 3, 5, …>, while array B contains n consecutive even numbers in decreasing order, i.e., B = <… 6, 4, 2>. Suppose you want to combine the contents of the two arrays and sort them in increasing order, i.e., the result is an array that contains <1, 2, 3, …, 2n>.

(i) **(9 marks).** Assume you want to solve the problem by first appending the contents of array B into array A, i.e., copying $B[1]$ to $A[n+1]$, $B[2]$ to $A[n+2]$, …, $B[n]$ to $A[n+n]$. Thus, A will contain <1, 3, 5, …, …, 6, 4, 2>. Then use a known sorting algorithm to sort array A that now contains $2n$ integers. Suppose you consider using the Quicksort and the Partition algorithms *provided at the end of this test paper*. **Answer each of the following questions.**

1) Show the result of running the Partition algorithm for $n = 5$, i.e., A = <1, 3, 5, 7, 9, 10, 8, 6, 4, 2>. **Note:** to answer this question, you are asked to use Partition only one time on the input.

2) Give the recurrence function for Quicksort when arrays A and B have n integers each. Explain your answer.

3) Give the time complexity of Quicksort for the input in part 2). Explain your answer. Note that you need not formally prove the time complexity.

(ii) **(3 marks).** If you use the Selection sort, for the solution in part (i), what is the time complexity of this algorithm? Explain your answer. **Note:** *the pseudocode for the Selection sort is provided at the end of this test paper*.

(iii) **(5 marks).** Consider now that you want to use the following MERGE() function to solve the problem. Explain how you modify the function to merge array A and array B such that the results are in increasing order. What is the time complexity of this solution? Justify your answer.

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 ≤ m) and (j ≤ r) do
    if A[i] ≤ A[j] then
        TEMP[k++] = A[i++]
    else
        TEMP[k++] = A[j++]
while (i ≤ m) do
    TEMP[k++] = A[i++]
while (j ≤ r) do
    TEMP[k++] = A[j++]
```

<u>Answer:</u>

(i) **Quicksort:**

1) Result of using Partition

2) Recurrence function

3) Time complexity

(ii) **Selection sort**

(iii) **MERGE** Modification

Time complexity:

b) **(Total: 7 marks).** The following Strassen's algorithm computes matrix product.

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

Strassen (*A*, *B*)

$n = rows[A]$
Let C be a new $n \times n$ matrix
if $n = 1$
    $c_{11} = a_{11} \times b_{11}$
else
$P_1 =$ **Strassen** $(A_{11}, B_{12} - B_{22})$
$P_2 =$ **Strassen** $(A_{11} + A_{12}, B_{22})$
$P_3 =$ **Strassen** $(A_{21} + A_{22}, B_{11})$
$P_4 =$ **Strassen** $(A_{22}, B_{21} - B_{11})$
$P_5 =$ **Strassen** $(A_{11} + A_{22}, B_{11} + B_{22})$
$P_6 =$ **Strassen** $(A_{12} - A_{22}, B_{21} + B_{22})$
$P_7 =$ **Strassen** $(A_{11} - A_{21}, B_{11} + B_{21})$
$C_{11} = P_5 + P_4 - P_2 + P_6$
$C_{12} = P_1 + P_2$
$C_{21} = P_3 + P_4$
$C_{22} = P_5 + P_1 - P_3 - P_7$
return C

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

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

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

(i) **(3 marks).** Complete the following partial computations of matrix product of the following using the Strassen's algorithm. Show your work.

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 1 & 5 \\ 2 & 4 \end{pmatrix}$$

$S_1 = 1$	$S_2 = 5$	$S_3 = 9$	$S_4 = 1$	$S_5 = 7$
$S_6 = 5$	$\boldsymbol{S_7 = ?}$	$S_8 = 6$	$S_9 = -2$	$S_{10} = 6$
$P_1 = 2$	$P_2 = 20$	$P_3 = 9$	$P_4 = 5$	$P_5 = 35$
$P_6 = -12$	$\boldsymbol{P_7 = ?}$			
$\boldsymbol{C_{11} = ?}$	$C_{12} = 22$	$C_{21} = 4$	$C_{22} = 40$	

(ii) **(4 marks).** Explain why the recurrence of the time complexity of the Strassen's algorithm is $T(n) = 7\, T(n/2) + \Theta(n^2)$.

Answer:

(i) Computation

$\boldsymbol{S_7 =}$

$\boldsymbol{P_7 =}$

$\boldsymbol{C_{11} =}$

(ii)

END OF QUESTION TWO

Attachment

Assume the following:
$lg3 \approx 1.5$, $lg5 \approx 2.3$, $lg6 \approx 2.5$, $lg7 \approx 2.8$, $lg9 \approx 3.1$, $lg10 \approx 3.3$.

Master Theorem:

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

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

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

```
if l < r
   then q ← ⌊(l+r )/2⌋
        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 ≤ m) and ( j ≤ r) do  // check if not at end of each sub-array
     if A[i] ≤ A[j] then              // check for smaller element
          TEMP[k++] = A[i++]
     else                         // copy smaller element
          TEMP[k++] = A[j++]          // into temp array
while (i ≤ m) do
     TEMP[k++] = A[i++]   // copy all other elements
while (j ≤ r) do            // to temp array
     TEMP[k++] = A[j++]
```

Quicksort(*A*,*l*,*r*)
Input : Unsorted Array (*A*,*l*,*r*); **Output :** Sorted array *A*(1..*r*)

```
if l < r
  then q ← 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 $A[i] \le A[m]$ for all $i \le m$ and $A[j] > A[m]$ for all $j > m$

```
x = A[r]
i = l
for j = l to r - 1 do
   if A[j] ≤ x then
         exchange A[i] ↔ A[j]
   i = i +1
exchange A[i] ↔ A[r]
return i
```

SELECTION_SORT (*A*[1, …, *n*])
Input: unsorted array *A* **Output:** sorted array *A*

```
for i ← 1 to n-1
   small ← i
   for j ← i+1 to n // Set small as the pointer to the  smallest element in A[i+1..n]
        if A[j] < A[small] then
             small ← j
   temp ← A[small]   // Swap A[i] and smallest
   A[small] ← A[i]
   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*)

```
let C [0 .. k] be a new array     // note that the index of array C starts from 0
for i = 0 to k
    C [i] = 0
for j = 1 to A.length
    C [ A [j] ] = C [ A [j] ] + 1
// C [i] now contains the number of elements equal to i
for i = 1 to k
    C [i ] = C [i] + C [i - 1]
// C [i] now contains the number of elements less than or equal to i
for j = A.length downto 1
    B [ C [ A [j] ]] = A [j]
    C [ A [j] ] = C [ A [j] ] - 1
```

END OF PAPER