CURTIN UNIVERSITY

DEPARTMENT OF COMPUTING

Test 1 – Semester 1 2017

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) **(5 marks).** Formally prove or disprove the correctness of the following. Note that you must show the values of c and n_0 to prove its correctness.

$2n^2 + 3n + 4 = \Theta(n^2)$

Answer:

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

$1^3 + 2^3 + 3^3 + \ldots + n^3 = (1 + 2 + 3 + \ldots + n)^2$

Note:
$(n + 1)^2 = n^2 + 2n + 1$

$(n + 1)^3 = n^3 + 3\,n^2 + 3n + 1$

Answer:

c) **(Total: 5 marks).** Consider the following pseudo code to answer the following questions.

```
i ← 0
while i < n do
    if ((A[i] MOD 2) == 0)  then
        return A[i]
    else
        i ← i + 1
```

(i) **(1 mark).** Briefly describe what the code does.

(ii) **(2 marks).** State the best-case of the algorithm and its best case asymptotic lower bound complexity, if there is any.

(iii) **(2 marks).** State the worst-case of the algorithm and its worst-case asymptotic upper bound complexity, if there is any.

Answer:

(i)

(ii)

(iii)

d) **(Total: 10 marks).**

(i) **(4 marks).** For recurrence $T(n) = 8T(n/2) + n^2$, is it possible to find the order of growth for the solution of recurrence using the Master theorem? If so, show the recurrence. Otherwise, show why the Master theorem cannot be used to produce its solution.

(ii) **(6 marks).** For recurrence $T(n) = 4T(n/2) + n^3$, use induction to prove that the solution to the recurrence is $O(n^3)$.

Answer:

(i) Master Theorem.

(ii) Induction:

END OF QUESTION ONE

QUESTION TWO (total: 24 marks).

a) **(Total: 10 marks).** Consider an application that requires maintaining a sorted array A of integers; assume they are sorted in increasing order. Regularly the application needs to inserts k numbers into the already sorted array A.

(i) **(4 marks).** One possible solution to keep array A sorted is to regularly sort the original elements in A together with the additional k numbers using any sorting algorithm. Is quicksort good for this solution? Explain your answer. What is the time complexity of quicksort for this case of input? Justify your answer.

(ii) **(6 marks).** For a constant k, design an algorithm for the problem that takes time $O(n)$, assuming the original size of array A is n. Write the pseudo code of the algorithm, and show that its time complexity is $O(n)$.

Note: If you use any algorithm that we have discussed in the lecture as part of your solution, you **need not** rewrite its pseudo code in your answer. However, you have to explain how you use it in your answer.

Answer:

(i)

(ii)

b) **(Total: 8 marks).** Consider the following algorithm to find the minimum difference between any two of its elements in an array of integers. For example, for A = [12, 10, 7, 1, 3, 6], the algorithm returns |7 – 6| = 1.

```
MinDifference_Ver1 (A[1 … n])

    d_min ← ∞

    for i ← 1 to n do
        for j ← 1 to n do // Line A
            if i ≠ j and |A[i] – A[j]| < d_min
                d_min ← |A[i] – A[j]|

    return d_min
```

(i) **(2 marks).** How many times will ***Line A*** be executed? Justify your answer.

(ii) **(1 mark).** State the worst-case upper bound time complexity of the algorithm. Note that you **need not** prove/justify your statement.

(iii) **(5 marks).** Write the pseudo code of one possible improved solution to the problem. Analyse the time complexity of your improved solution.

Note: An improvement can be either in terms of **smaller time complexity** or **reduced total number of steps**. Thus your answer has to give the improved time complexity or improved total number of steps.

Answer:

(i)

(ii)

(iii)

c) **(Total: 6 marks).** You are asked to compute a^n, for a power of 2 positive integer n, e.g., for $a = 5$ and $n = 8$ (a power of 2, i.e., 2^3), compute $5^8 = 390625$.

Note that $n = 6$ is not a power of 2. **Hint.** $5^8 = 5^4 * 5^4$

(i) **(4 marks).** Write the pseudo code for a divide and conquer algorithm to compute a^n.

(ii) **(2 marks).** Give the recurrence function for the number of multiplications made by this algorithm. Explain your answer.

Answer:

(i) pseudocode

(ii) recurrence

END OF QUESTION TWO

Attachment

Master Theorem:

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

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

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 = 0
j = m+1
k = 0
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 subarray *A*(0..*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*] ≤ *A*[*m*] for all *i* ≤ *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*
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*

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.

END OF PAPER