Venue _____

Student Number |__|__|__|__|__|__|__|__|

Family Name _____

First Name _____

End of Semester 1, 2019
COMP3001 Design and Analysis of Algorithms

Curtin University

School of Electrical Engineering, Computing and Mathematical Sciences

EXAMINATION

End of Semester 1, 2019

COMP3001 Design and Analysis of Algorithms

This paper is for Bentley Campus students

This is a CLOSED BOOK examination

Examination paper IS NOT to be released to student

Examination Duration 2 hours

Reading Time 10 minutes

Students may write notes in the margins of the exam paper during reading time

Total Marks 100

Supplied by the University

None

Supplied by the Student

Materials

None

Calculator

No calculators are permitted in this exam

Instructions to Students

ATTEMPT ALL QUESTIONS IN THE SPACE PROVIDED

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

Total _____

QUESTION ONE (Total: 24 marks).

a) **(2 marks).** Rank the following functions by increasing order of their growth rate.

- $2n \log n$
- $5n^{0.5} \log n$
- 2^{100}

Answer:

b) **(4 marks).** Consider the following code where A takes $O(n^{1.5})$ time and B takes $O(n)$ time.

```
If (X(n)) then
    A
else
    B
```

If $X(n)$ is a function with running time complexity $O(n \log n)$, what are the asymptotic upper bound on the *best case* and *worst case* running time of the code?

Answer:

c) **(Total: 12 marks).** Consider the pseudo code for function **A** to multiply the values of integers x and y.

```
A (x, y)
   result ← 0;
   repeat
      if x is odd then
            result ← result + y
      x = x / 2   // this is an integer division
      y = y + y
   until x == 1
   return result
```

(i) **(5 marks).** Given $x = 25$ and $y = 50$, trace the pseudo code for the given input. You have to show the value of x and y for each repetition, and the value of *result* when the algorithm terminates.

(ii) **(5 marks).** Analyse the running time complexity of the algorithm in terms of the value of x. In your analysis, give the number of steps for each line of the code. Finally, show the complexity in its asymptotic upper bound. **You are not asked to provide any proof.**

(iii) **(2 marks).** Suppose you want to multiply two numbers, 109872 and 23, using the algorithm. Which of the two numbers would you set as x so that the algorithm runs faster? Why?

Answer:

(i)

(ii) Time complexity $O(\lg x)$

(iii)

d) **(Total: 6 marks).** Show using either the Master method or induction that the asymptotic upper bound time complexity of each of the following recurrence functions is as given. **Hint.** Assume $\log_{10} 3 = 0.5$; $\log 2 = 0.3$.

(i) **(3 marks).** $T(n) = 3\ T(2n/3) + n \rightarrow O(n^{2.71})$

(ii) **(3 marks).** $T(n) = T(n-1) + n \rightarrow O(n^2)$

Answer:

(i)

(ii)

END OF QUESTION ONE

QUESTION TWO (Total: 24 marks).

a) **(Total: 12 marks).**

Consider the following algorithm that has as input an n-element array A of numbers and gives as output an n-element array B of numbers such that $B[i]$ is the average of elements $A[0], \ldots, A[i-1]$. For input $A = (1, 2, 3, 4, 5, 6, 7)$, the algorithm produces $B = (1, 3/2, 6/2, 10/4, 15/5, 21/6, 28/7)$.

```
1. for i ← 0 to n – 1  do
2.     b ← 0
3.     for j ← 0 to i do
4.         b ← b + A[j]
5.         B[i] ← b / (i + 1)
6. return array B
```

(i) **(4 marks).** Show that the time complexity of the algorithm is $O(n^2)$. You need to give the number of steps needed for each line of the algorithm, and sum them up to compute the complexity.

(ii) **(8 marks).** Modify the algorithm into an $O(n)$ algorithm.

- Show your algorithm in a pseudo code.
- Show that your algorithm is in $O(n)$.
- Use the input $A = (1, 2, 3, 4, 5, 6, 7)$ to show how your algorithm works.

Answer:

(i)

(ii)

b) **(10 marks).** Consider a cashier in a bank who serves n customers. Assume customer i needs service time t_i, $1 \leq i \leq n$, and we want to schedule the customers to minimize the average time that a customer spends in the bank. As an example, for $t_1 = 5$, $t_2 = 10$, and $t_3 = 3$, if the customers are scheduled in order of t_1, t_2, and t_3, the average time that a customer spends in the bank is calculated as $\{5 + (5+10) + (5+10+3)\} / 3 = 38/3$, which is NOT the minimum average time.

(i) **(1 mark).** Compute the minimum average time for the given example.

(ii) **(5 marks).** Design a greedy algorithm for this problem.

- You have to state your greedy choice.
- Use the binary heap in your algorithm.

Note: You are not required to present your algorithm in a pseudo code. You are allowed to present the algorithm in steps using clear sentences.

(iii) **(4 marks).** Analyse the time complexity of your algorithm.

Answer:

(i)

(ii)

(iii)

c) **(2 marks).** List one algorithm discussed in DAA that uses each of the following algorithm design paradigm. **Note: 1 mark will be deducted for each wrong answer, to the minimum of zero mark for this question.**

- Dynamic programming.
- Divide and conquer.
- Greedy.

Answer:

- **Dynamic programming:**

- **Divide and conquer:**

- **Greedy:**

END OF QUESTION TWO

QUESTION THREE (Total: 20 marks).

a) **(Total: 8 marks).** Consider the following Rabin-Karp string matcher algorithm.

RABIN-KARP-MATCHER(*T, P, d, q*)
Input: Text *T*, pattern *P*, radix *d* (which is typically = $|\Sigma|$), and the prime *q*.
Output: valid shifts s where *P* matches

1. $n \leftarrow length[T]$
2. $m \leftarrow length[P]$
3. $h \leftarrow d^{m-1} \bmod q$
4. $p \leftarrow 0$
5. $t_0 \leftarrow 0$
6. **for** $i \leftarrow 1$ to m
7. **do** $p \leftarrow (d*p + P[i]) \bmod q$
8. $t_0 \leftarrow (d*t_0 + T[i]) \bmod q$
9. **for** $s \leftarrow 0$ to $n-m$
10. **do if** $p = t_s$
11. **then if** $P[1..m] = T[s+1..s+m]$
12. **then** "pattern occurs with shift s "
13. **if** $s < n-m$
14. **then** $t_{s+1} \leftarrow (d*(t_s - T[s+1]*h) + T[s+m+1]) \bmod q$

(i) **(4 marks).** Explain why the time complexity of the algorithm for the *best case* and the *worst case* scenarios are $\Theta((n - m + 1))$ and $\Theta(m\,(n - m + 1))$, respectively.

(ii) **(2 marks).** For $T = 1242345$, $P = 34$, $d=10$, and $q = 11$, compute t_0, t_2. **Hint:** you are NOT required to trace the algorithm for your answer.

(iii) **(2 marks).** For the example in part (ii), how many t_s (including t_0) are computed? How many spurious hits are there?

Answer:

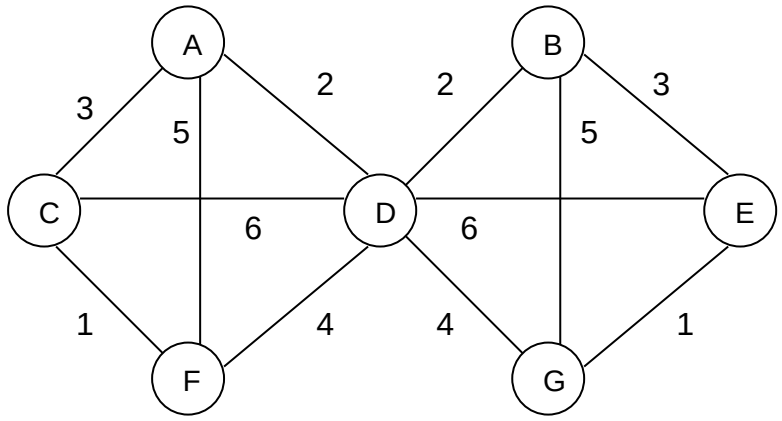
(i) **Best case**:

Worst case:

(ii)

(iii)

b) **(Total: 12 marks).** Consider the following weighted graph.



(i) **(3 marks).** Show the adjacency list representation of the given weighted graph.

(ii) **(7 marks).** Use the following Prim's algorithm to generate the MCST of the given graph. Show the resulting MCST and its cost. **When there is a tie, follow the alphabetical order.**

1. Insert all $v \in V$ into a priority queue Q, each with key of *infinity*.
   // key[v] is the weight of the edge connecting v to the MCST
2. **while** Q is not empty **do**
3. Take out u, the min key vertex, from Q
4. **for** each edge (u, v) **do**
5. **if** weight of (u, v) < key[v] **then**
6. Record u as parent of v
7. Decrease key to weight of (u, v)
8. The final tree is the set of edges { (u, parent[u]) }

(iii) **(2 marks).** Give one main difference and one main similarity between Prim's algorithm and Kruskal's algorithm in constructing their MCST.

Answer:

(i) Adjacency list

(ii) MCST using Prim's algorithm

(iii)

END OF QUESTION THREE

QUESTION FOUR (Total: 32 marks).

a) **(Total: 12 marks)**. Consider the following dynamic programming algorithm to solve the matrix chain problem, and its resulting table *m* and *s* given four consecutive matrices as input: $A = 20\times10$, $B = 10\times50$, $C = 50\times5$, and $D = 5\times30$.

Input: sequence $P = (p_0, p_1, \ldots p_n)$.
Output: an auxiliary table $m[1..n, 1..n]$ with $m[i, j]$ costs and another auxiliary table $s[1..n, 1..n]$ with records of index k which achieves optimal cost in computing $m[i, j]$.

Matrix_Chain_Order (*P*).

```
1.  n ←length[P] – 1;
2.  for i ← 1 to n
3.      do m[i, i] ← 0;
4.  for l ← 2 to n
5.      do for i ← 1 to n- l +1
6.          do j ← i + l -1
7.              m[i, j] ←∝;
8.              for k ← i to j -1
9.                  do q ← m[i, k] + m[k +1, j] + p_{i-1} p_k p_j;
10.                     if q < m[i, j]
11.                         then m[i, j] ← q;
12.                              s[i, j] ← k;
13. return m and s
```

m

j \ *i*	1	2	3	4
1	0			
2	10000	0		
3	3500	2500	0	
4	6500	4000	7500	0

s

j \ *i*	1	2	3	4
1				
2	1			
3	1	2		
4	3	3	3	

(i) **(2 marks).** What are the contents of *P* as the input to the algorithm that produces table *m* and *s*?

(ii) **(4 marks).** Show how the algorithm computes $m[1, 4] = 6500$. **NOTE:** (1) You have to give the algorithm's detailed calculation; (2) If you know the algorithm, to save time you need not trace the algorithm to answer the question.

(iii) **(2 marks).** For the given solution, show the optimal bracketing that gives the minimum number of multiplications.

(iv) **(4 marks).** Suppose you want to solve the problem using a greedy approach. What would be your **greedy choice** for your approach? Give the solution for the given sequence of matrices using your greedy approach. **NOTE:** you are not asked to write any pseudo code.

Answer:

(i)

(ii)

(iii)

(iv)

b) **(Total: 10 marks).** Consider the following two alternative recursive functions to solve the 0/1 knapsack problem.

```
/*************************************************/
KNAPSACK-RECURSE_V1 (i, k)      // VERSION 1
  if (i = n) then
     if (w_n > k) then
       return 0
     else
       return p_n

  if (w_i > k) then
     return KNAPSACK-RECURSE(i +1, k)
  else
     x := KNAPSACK-RECURSE (i + 1, k)
     y := KNAPSACK-RECURSE (i + 1, k – w_i) + p_i
     return max(x, y)

/*************************************************/
KNAPSACK-RECURSE_V2 (i, k)  // VERSION 2
  // Note: P is an array of size C + 1, i.e., P[0 .. C]
  if P[k] != UNKNOWN then
       return P[k]

  if (i = n) then
      if (w_n > k) then
         return 0
      else
         return p_n

  if (w_i > k) then
      return KNAPSACK-RECURSE (i+1, k)
  else
       x = KNAPSACK-RECURSE (i+1, k)
       y = KNAPSACK-RECURSE (i+1, k–w_i)
       P[k] = max (x, y)
       return P[k]
/*************************************************/
```

(i) **(2 marks).** State the recurrence function of the time complexity of the recursive algorithm **VERSION 1.** Explain your answer.

(ii) **(2 marks).** State the upper bound time complexity of **VERSION 1.** Briefly explain your answer. **NOTE:** you are NOT asked to give a formal proof. A short statement is sufficient.

(iii) **(3 marks).** Explain why **VERSION 2** has an upper bound time complexity of $O(C*n)$, where C is the total weight constraint.

(iv) **(3 marks).** The 0/1 aims to find a set of selected items that gives the maximum profit. Explain if **VERSION 2** generates such set. If not, explain how you modify the algorithm to generate such set.
NOTE: you are NOT asked to write any pseudo code as your answer. A brief explanation is sufficient for your answer.

Answer:

(i)

(ii)

(iii)

(iv)

c) **(Total: 10 marks).** Consider the following parallel search algorithm to determine if the array *A* contains *x*.

```
/*********************************************/
Parallel_Search_EREW (x, A[1 .. n])

Broadcast(x, B[1 .. n])

forall Pi do in parallel
    if A[i] = B[i] then
        B[i] ← i
    else
        B[i] ← ∞
endfor

return i = Fan_in (B[1 .. n])

/*********************************************/
Broadcast (x, B[1 .. n])

B[1] = x
for i = 1 to log n do
   forall Pk where 2^(i-1) ≤ j ≤ 2^i-1 and k = j + 1 - 2^(i-1) do in parallel
      B[j+1] ← B[j + 1 - 2^(i-1)]
   endfor
endfor

/*********************************************/
Fan_in (B[1 .. n])

for i = 1 to log n do
     forall Pj where 1 ≤ j ≤ n/2 do in parallel
        if 2j mod 2^i = 0 then
           if B[2j] > B[2j – 2^(i-1)] then
              B[2j] ← B[2j – 2^(i-1)]
/*********************************************/
```

(ii) **(2 marks).** Explain why the parallel search algorithm needs to use function *Broadcast*(.).

(iii) **(4 marks).** Is function *Broadcast*(.) **cost optimal**? Explain your answer.

(iv) **(4 marks).** If the cost function *Fan_in*(.) is $O(n\log_2 n)$, is the parallel search algorithm **cost efficient**? Explain your answer.

Answer:

(i)

(ii)

(iii)

END OF QUESTION FOUR

Attachment

Assume the following:

$log_{10}3 \approx 0.5$, $log_{10}2 \approx 0.3$, $lg3 \approx 1.5$, $lg5 \approx 2.3$, $lg6 \approx 2.5$, $lg7 \approx 2.8$, $lg9 \approx 3.1$, $lg10 \approx 3.3$.

$\log(a/b) = \log a - \log b$

$\log(ab) = \log a + \log b$

Master Theorem:

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

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

END OF EXAMINATION