Curtin University

School of Electrical Engineering, Computing and Mathematical Sciences

Discipline of Computing - Curtin University

COMP3001 Design and Analysis of Algorithms

Final Assessment - Semester 1/2021

Total Mark: 100

Time allowed: 4 hours (start to finish)

Assessment Availability: 8.30am - 12.30pm, Thursday 10th of June 2021

Test mode: Online test, open-book: you are allowed to access your hand-written notes, lecture slides, textbooks, and printed and electronic materials in your possession. **However, you are NOT allowed to simply copy and paste solutions from your open sources. Doing so will be considered academic misconduct.**

CONDITIONS

- The assessment must be completed by yourself only. No one else should do this assessment for you. Any attempts to compromise the system are strictly prohibited. Any breaches of this policy will be considered cheating and appropriate action will be taken as per University policy.
- You are prohibited from communicating with people other than the unit coordinator/lecturer and the tutors during the assessment.
- You are prohibited from providing information about your work and your assessment to others during and outside your assessment within two days. Some students may sit the assessment after you.
- You must complete, sign and submit the "Student Declaration Form".
- Some students sitting the assessment can be invited to an online interview as confirmation check to confirm the authorship of the submitted final assessment. In the interview, students will be asked to explain their answers and demonstrate their knowledge for randomly selected questions. Students will be shown the questions, as well as their written answers.

INSTRUCTION

- This assessment consists of four questions: QUESTION ONE to QUESTION FOUR of different types, with a total of 100 marks. **Attempt ALL questions**.
- You can submit your answers multiple times during the assessment, but only the last submission would be used for marking.
- You are allowed to write or type your answers. Submit the pdf file of your answers.
- When you type your answers, you are allowed to 1) use ^ to represent a superscript and _ to represent a subscript; e.g., n^2 for n^2 and n_2 for n_2; 2) use floor(x) and ceiling(x) to represent the floor and ceiling functions of a value x, respectively, e.g., floor(1.9) for $\lfloor 1.9 \rfloor$, and ceiling(1.1) for $\lceil 1.1 \rceil$.

QUESTION ONE (Total: 20 marks).

a) **(10 marks).** It is known that no comparison-based sorting algorithm has a running time complexity less than $\Omega(n \lg n)$.

- Suppose your friend *A* claims to have a comparison-based sorting algorithm that satisfies a recurrence $T(n) = 5T(n/6) + n$. Is your friend's claim reasonable? If so, would you use your friend's algorithm over merge sort for $n \leq 10000$? Ignore the constant values in time complexities. Justify your answer.

- Suppose your friend *B* claims to have a comparison-based sorting algorithm that satisfies a recurrence $T(n) = 4T(n/3) + n$. Is your friend's claim reasonable? If so, would you use your friend's algorithm over merge sort for $n \leq 10000$? Ignore the constant values in time complexities. Justify your answer.

Answer:

- Algorithm A. $T(n) = 5T(n/6) + n$.

- Algorithm B. $T(n) = 4T(n/3) + n$.

b) **(6 marks).** The time complexity of the quicksort to sort n elements is $O(n \lg n)$ if the partition function produces split with constant proportionality, for example, $2n/10$ of the elements are on the left side of the selected pivot and $8n/10$ of the elements in its right side. In this scenario, the recurrence relation of the quicksort's time complexity is $T(n) = T(2n/10) + T(8n/10) + n$. Use induction to prove the time complexity $T(n)$ is $O(n \lg n)$.

Answer:

c) **(4 marks).** Give a recurrence for the running time of the following algorithm ABCD, where A is an array of size n, i is the leftmost index of array A, and j is the rightmost index. Briefly explain why your relation is correct.

```
ABCD (A, i, j)
1   if A[i] > A[j] then
2         exchange A[i] ↔A[j]
3   if i+1 ≥ j then
4         return
5   k ← ⌊(j-i+1)/3⌋
6   ABCD (A, i, j-k)
7   ABCD (A, i+k, j)
8   ABCD (A, i, j-k)
```

Note: You do not need to understand the ABCD algorithm to answer the question.

Answer:

END OF QUESTION ONE

QUESTION TWO (Total: 20 marks).

a) **(Total: 8 marks).** Consider an array A that contains n integers. You are asked to design an algorithm to find two integers in A that give the smallest sum. Example A = <2, 4, 6, -1, 5, 5> will result in 2 + (-1) = 1. Thus, the two integers are 2 and -1.

(i) **(3 marks).** Write the pseudocode code of your algorithm with a time complexity no worse than $O(n \lg n)$. You have to show that the time complexity of your algorithm is $O(n \lg n)$.

(ii) **(5 marks).** Write the pseudocode of an algorithm with time complexity of $O(n)$ for the problem. You have to show that the time complexity of your algorithm is $O(n)$.

Answer:

(i)

(ii)

b) **(Total: 12 marks).** Consider a set X of n nuts with possibly k different sizes, and another set Y of n bolts with possibly k different sizes, where k is a constant. A nut matches a bolt if they have the same size. Assume the size is in integers. For example, for $k = 5$, and $n = 10$, we can have nuts = <2,1,3,5,4,1,1,4,3,4> and bolts = <1,1,2,3,4,2,1,3,5,4>, where each number represents a nut or bolt of that size.

(i) **(5 marks).** Write the pseudocode of an $O(n \lg n)$ algorithm to find the total number of all matched nuts and bolts (i.e., the total number of pairs (a, b) such that for $a \in X$, and $b \in Y$, a and b have the same size). For example, for nuts = <2,1,3,5,4,1,1,4,3,4> and bolts = <1,1,2,3,4,2,1,3,5,4>, the output is 9. You have to show that your algorithm is $O(n \lg n)$.

(ii) **(5 marks).** Is it possible to use the LCS-Length (X, Y) function to solve the problem in part (i)? i.e., to find the total number of matched nuts and bolts.
If it is possible, design an algorithm that uses LCS-Length (X, Y) to solve the problem. Analyse the time complexity of this algorithm. You can write your algorithm in steps. Further, you don't need to rewrite the list of code in LCS-Length (X, Y) in your answer, i.e., you can assume that the function returns the longest length of LCS between string X and Y.
If it is not possible, clearly explain the reasons.

Note: the LCS-Length (X, Y) is a dynamic programming algorithm to find the longest common subsequence between two strings X and Y.

(iii) **(2 marks).** Is it possible to design an algorithm with time complexity of $O(\lg n)$ to solve the problem in part (i)? i.e., to find the total number of matched nuts and bolts? Explain your answer.

Answer:

(i)

(ii) Using LCS.

(iii)

END OF QUESTION TWO

QUESTION THREE (Total: 26 marks).

a) **(Total: 7 marks).** A file has alphabets {*A, B, C, D, E*}, where *A* has frequency 5, *B* has frequency 15, *C* has frequency 6, *D* has frequency 14, and *E* has frequency 10.

(i) **(3 marks).** Show the code tree and the Huffman code for each of the alphabets.

(ii) **(4 marks).** Show that the computed Huffman code is optimal.

Answer:

(i) Code tree and Huffman code

(ii)

b) **(Total: 7 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) **(2 marks).** Suppose we remove Line 13. Is the algorithm still correct? Explain your answer.

(ii) **(3 marks).** For T = 123456789123456789, P = 34, d=10, q = 11, and t_1 = 1, trace the algorithm to compute t_2.

(iii) **(2 marks).** T = 123456789123456789, P = 34, d=10, q = 11, how many t_s (including t_0) are computed? How many spurious hits are there?

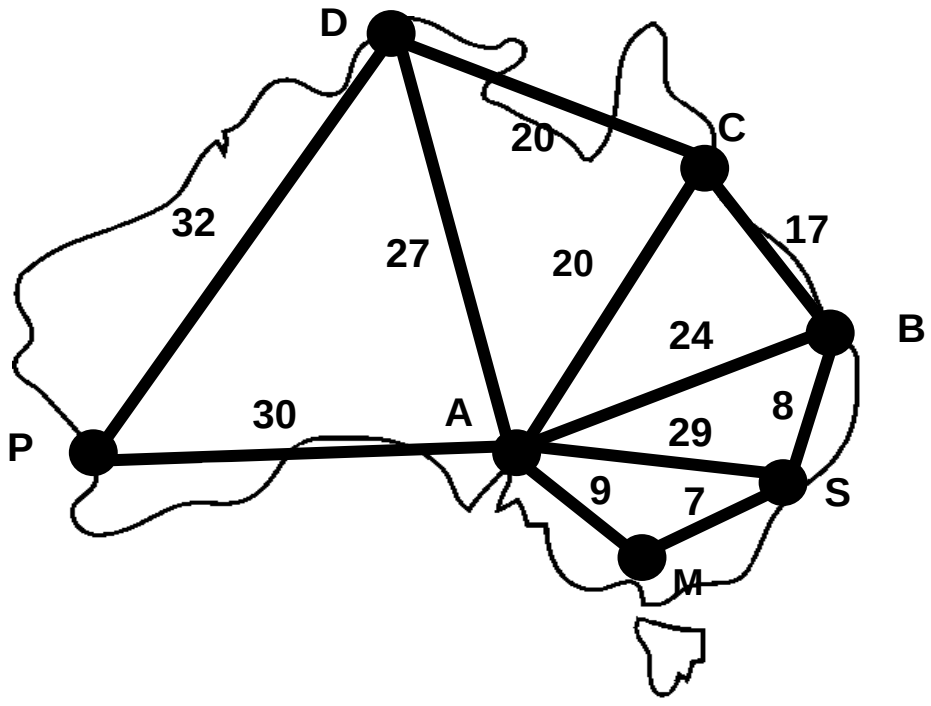
Answer:

(i)

(ii)

(iii)

c) **(Total: 12 marks**). Find the cheapest way (in Dollar value) you can send a message to all seven routers located in cities marked on the following map. Each router is allowed to copy the message and pass it on to its neighbour router, whilst retaining a copy for itself. The cost of sending a message between two routers is directly proportional to the distance between the routers, which are shown on the map. Assume it costs $0.50 to transmit the message for each unit distance. Note that the message might originate in any one of the routers.



(i) **(2 marks).** What graph problem that we have studied is the best way to model this problem? What algorithm that we have studied would you use to find the optimal route?

(ii) **(10 marks).** Show the detail workings of your algorithm chosen in part (i) and the optimal solution (in Dollar value).

Answer:

(i)

(ii)

END OF QUESTION THREE

QUESTION FOUR (Total: 34 marks).

a) **(Total: 15 marks).** Suppose that in a 0/1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing profit. For example, for five items, we can have w = [1, 2, 3, 4, 5] and p = [5, 4, 3, 2, 1], or w = [1, 2, 4, 5, 7] and p = [8, 6, 5, 3, 1].

(i) **(6 marks).** Design an efficient algorithm to find the maximum profit for this variant of the 0/1 knapsack problem.
- Write the pseudocode of your algorithm.
- Show how your algorithm produces the maximum profit for input w = [1, 2, 4, 5, 7] and p = [8, 6, 5, 3, 1] for capacity c = 8.
- Analyse the running time complexity of your solution for n items and capacity c.

(ii) **(4 marks).** Use the following dynamic algorithm (from the lecture) to produce output for input w = [1, 2, 4, 5, 7] and p = [8, 6, 5, 3, 1].

Knapsack (*S*,*C*)
Input: Set S of n items with p_i profit and w_i weight, and maximum weight C
Output: profit $P[w]$ of a subset S with total weight at most w, for w= 0, 1, … C

```
for k ← 0 to C do
    P[k] ← 0
for i ← n downto 1 do  // Line A
    for k ← C downto w_i do  // Line B
        if P[k - w_i] + p_i > P[k] then
            P[k] ← P[k - w_i] + p_i
```

(iii) **(5 marks).** Modify the Knapsack (S, C) algorithm from simply computing the best profit value to computing the item selection that gives this profit. You have to describe how to use your modified algorithm to get selected items.

Answer:

(i)

(ii) Dynamic programming:

Item/k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
5									
4									
3									
2									
1									

(iii) Modification:

b) **(Total: 13 marks)** Consider a sequence P = (5, 3, 1, 4, 6) that specifies a sequence of matrices, A, B, C, and D.

(i) **(1 mark).** List the size of each of the matrices.

(ii) **(8 marks).** Use the following dynamic algorithm to
- Fill out table *m* and *s*.
- Write in detail how to compute entry *m*[1,4] and entry *s*[1,4].
- Determine the optimal ordering of the matrix multiplications, and compute the optimal number of scalar multiplications.

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

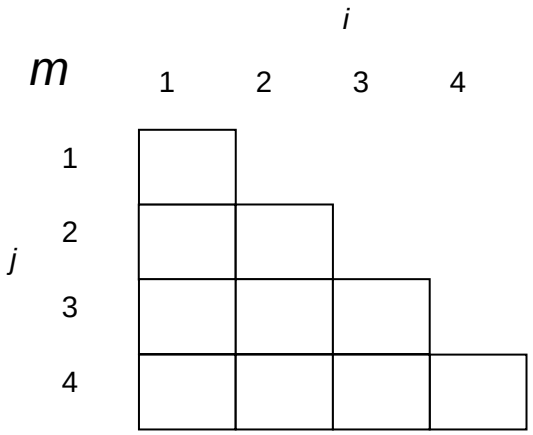
```
1.  n ←length[p]-1;
2.  for i ← 1 to n
3.      do m[i, i] ← 0;
4.  for l ← 2 to n
5.      do for i ← 1 to n- l +1
6.          do j ← i + l -1
7.              m[i, j] ←∝;
8.              for k ← i to j -1
9.                  do q ← m[i, k]+ m[k +1, j]+ p_{i-1} p_k p_j;
10.                     if q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

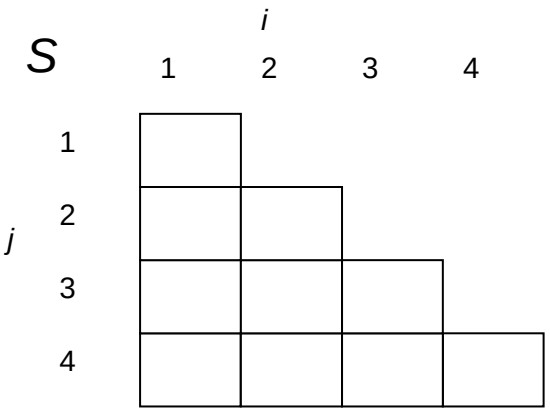
(iii) **(4 marks).** If you solve the problem using a greedy approach, how many scalar multiplications are required to multiply the matrices? State your greedy criteria. Is your greedy approach optimal? Justify your answer.

Answer:

(i)

(ii) Table *m* and *s*





(iii)

c) **(Total: 6 marks).** Consider the following parallel search algorithm.

```
Algorithm Parallel_Search (x, A[1 .. n])
index ← -1    // initialized with an invalid index value
forall Pi do in parallel    // 1 ≤ i ≤ n
    if A[i] = x then
        index ← i
    endif
endfor
```

(i) **(4 marks).** Modify the algorithm such that it uses only $n/2$ processors.

(ii) **(2 marks).** Is the algorithm in part (i) cost optimal? Explain your answer.

Answer:

(i) Modification

(ii)

END OF QUESTION FOUR

EXTRA SPACE FOR YOUR ANSWERS, IF NEEDED

END OF EXAMINATION