THE UNIVERSITY OF SYDNEY

CONFIDENTIAL EXAM PAPER

Computer Science

EXAMINATION

Semester 1- Practice, 2022

COMP2823 Data Structures and Algorithms (Adv)

EXAM WRITING TIME: 2 hours
READING TIME: 10 mins

EXAM CONDITIONS:
This is a OPEN book examination.
All submitted work must be **done individually** without consulting someone else's help, in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

MATERIALS PERMITTED IN THE EXAM VENUE:

MATERIALS TO BE SUPPLIED TO STUDENTS:

INSTRUCTIONS TO STUDENTS:
Type your answers in your text editor (Latex, Word, etc.) and convert it into a pdf file.

Submit this pdf file via Canvas. No other file format will be accepted. Hand-written responses will **not** be accepted.

Start by typing you student ID at the top of the first page of your submission. Do **not** type your name.

Submit only your answers to the questions. Do **not** copy the questions.

Do **not** copy any text from the permitted materials. Always write your answers in your own words.

For examiner use only:

Problem	1	2	3	4	5	Total
Marks						
Out of	10	10	15	15	10	60

Problem 1.

a) Analyze the time complexity of this algorithm. [5 marks]

```
def COMPUTE(A)
    result ← 0
    for i = 0; i < n; i++ do
        if A[i] > i then
            result ← result + A[i]
    return result
```

b) We are planning a board games event and we're using one of the shelves in my office to store the games. Unfortunately the shelf only has a certain amount of space S, so we need to carefully pick which games we want to bring. Every game takes some space s_i and has a fun factor f_i that indicates how much fun it is to play that game (for $1 \leq i \leq n$). [5 marks]

We want to maximize the amount of fun we'll have, so we want to maximize the sum of the fun factors of the games we pick (i.e., max $\sum_{\text{picked game } i} f_i$), while making sure that the games fit on my shelf, so the sum of the space the games we pick take should be at most S (i.e., $\sum_{\text{picked game } i} s_i \leq S$). For simplicity, you can assume that all f_i, s_i, and S are all distinct positive integers.

The strategy of PICKLARGEST is to always pick the game with the highest fun factor until my shelf is full: it sorts the games by their fun factor f_i in decreasing order and adds a game when its required space is less than the remaining space on the shelf.

```
def PICKLARGEST(all f_i and s_i, S)
    currentSpace ← 0
    currentFun ← 0
    Sort games by f_i and renumber such that f_1 ≥ f_2 ≥ ... ≥ f_n
    for i ← 1; i ≤ n; i++ do
        if currentSpace + s_i ≤ S then
            currentSpace ← currentSpace + s_i          ▷ Pick the ith game
            currentFun ← currentFun + f_i
    return currentFun
```

Show that PICKLARGEST doesn't always return the correct solution by giving a counterexample.

Problem 2. The product of two $n \times n$ matrices X and Y is a third $n \times n$ matrix $Z = XY$, where the (i, j) entry of Z is $Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$. This definition immediately leads to an $O(n^3)$ time algorithm for matrix multiplication. Here we explore the option of designing an alternative algorithm using divide and conquer. Suppose that X and Y are divided into four $n/2 \times n/2$ blocks each:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Using this block notation we can express the product of X and Y as follows

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

In this way, one multiplication of $n \times n$ matrices can be expressed in terms of 8 multiplications and 4 additions that involve $n/2 \times n/2$ matrices. It is straightforward to translate this insight into a divide and conquer algorithm for matrix multiplication; unfortunately, this new algorithm's time complexity is again $O(n^3)$.

Suppose that instead of 8 recursive multiplications of $n/2 \times n/2$ matrices, we could compute the product using only 4 such matrix multiplications and a constant number of matrix addition operations. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this improved recursive algorithm. **Your task** is to

a) Derive the recurrence for $T(n)$. (Assume that adding two $k \times k$ matrices takes $O(k^2)$ time.) [5 marks]

b) Solve the recurrence by unrolling it or using the Master Theorem. [5 marks]

Problem 3. Consider the *Dynamic Matrix* ADT for representing an matrix $A = \{a_{i,j}\}_{i,j=1}^{n}$ that supports the following operations:

- CREATE(): creates a 1×1 matrix where $a_{1,1} = 0$.
- SET/GET(i, j): set or get the value of the entry $a_{i,j}$.
- INCREASE-SIZE: If the current size of the matrix is $n \times n$, increase it to $n+1 \times n+1$ such that the new entries are set of 0. In other words, A becomes A' such that $a'_{i,j} = a_{i,j}$ if $1 \leq i, j \leq n$, and $a'_{i,j} = 0$ otherwise.

Your task is to come up with a data structure implementation for the Dynamic Matrix ADT that uses $O(n^2)$ space, where n is the size of the matrix, and CREATE, SET, GET take $O(1)$ and INCREASE-SIZE takes $O(n)$ time. Remember to:

a) Describe your data structure implementation in plain English. [6 marks]

b) Prove the correctness of your data structure. [6 marks]

c) Analyze the time and space complexity of your data structure. [3 marks]

Problem 4. Let G be a connected undirected graph on n vertices. We say that two distinct spanning trees T and S of G are *one swap away* from each other if $|T \cap S| = n - 2$; that is, T and S differ in only one edge.

For two distinct spanning trees T and S we say that $R_1, R_2, \ldots, R_k$ form a *swapping sequence* from T to S if:

1. $R_1 = T$,
2. $R_k = S$, and
3. for any $1 \leq i < k$, the trees R_i and R_{i+1} are one swap away from each other

Your task is to design a polynomial time algorithm that given G and two spanning trees T and S of G, constructs a minimum length swapping sequence. Remember to:

a) Describe your algorithm in plain English. [6 marks]

b) Prove the correctness of your algorithm. [6 marks]

c) Analyze the time complexity of your algorithm. [3 marks]

Problem 5. One last hard problem here. [10 marks]