



THE UNIVERSITY OF
SYDNEY

Student number: _____

Room number: _____

Seat number: _____

ANONYMOUSLY MARKED

(Please do not write your name on this exam paper)

CONFIDENTIAL EXAM PAPER

This paper is not to be removed from the exam venue.

Computer Science

EXAMINATION

Semester 1- Practice, 2023

COMP2X23 Data Structures and Algorithms

EXAM WRITING TIME: 150 minutes

READING TIME: 10 minutes

EXAM CONDITIONS:

This is an OPEN book examination.

Writing is not permitted during reading time.

MATERIALS PERMITTED IN THE EXAM VENUE:

Only printed or handwritten material is permitted.

(No electronic devices of any kind are permitted.)

MATERIALS TO BE SUPPLIED TO STUDENTS:

This exam paper booklet.

INSTRUCTIONS TO STUDENTS:

Write your student ID at the top right of this front page.

Write your answers in the spaces provided in the exam paper booklet using blue or black ink. If you run out of space continue on the blank pages at the end of the exam paper.

Please tick the box to confirm that your examination paper is complete. ☐

For examiner use only:

Problem	1	2	3	4	5	Total
Marks						
Out of	20	20	20	20	20	100

Problem 1.

a) Analyze the time complexity of this algorithm.

```
1: def COMPUTE(A)  
2:   result  $\leftarrow$  0  
3:   for i = 0; i < n; i ++ do  
4:     if A[i] > i then  
5:       result  $\leftarrow$  result + A[i]  
6:   return result
```

b) Solve the following recursion using unrolling:

$$T(n) = \begin{cases} T(n/2) + O(1) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

(solution to Problem 1 continues here)

Problem 2. 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 $1 \leq i \leq n$).

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.

```
1: def PICKLARGEST(all  $f_i$  and  $s_i$ ,  $S$ )
2:    $currentSpace \leftarrow 0$ 
3:    $currentFun \leftarrow 0$ 
4:   Sort games by  $f_i$  and renumber such that  $f_1 \geq f_2 \geq \dots \geq f_n$ 
5:   for  $i \leftarrow 1$ ;  $i \leq n$ ;  $i++$  do
6:     if  $currentSpace + s_i \leq S$  then
7:        $currentSpace \leftarrow currentSpace + s_i$             $\triangleright$  Pick the  $i$ th game
8:        $currentFun \leftarrow currentFun + f_i$ 
9:   return  $currentFun$ 
```

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

(solution to Problem 2 continues here)

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.
- b) Prove the correctness of your data structure.
- c) Analyze the time and space complexity of your data structure.

(solution to Problem 3 continues here)

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, \dots, 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.
- b) Prove the correctness of your algorithm.
- c) Analyze the time complexity of your algorithm.

(solution to Problem 4 continues here)

Problem 5. (Description of hard problem here)

(solution to Problem 5 continues here)

(exam questions end here. use the rest of the space for rough work)

(use the rest of the space of rough work)

(use the rest of the space of rough work)

(use the rest of the space of rough work)

(use the rest of the space of rough work)