

National University of Singapore
School of Computing

CS3230 - Design and Analysis of Algorithms
Final Assessment
(Semester 2 AY2022/23)

Time Allowed: 2 hours

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains **TWO** (2) sections.
It comprises **FOURTEEN** (14) printed pages, including this page.
3. This is an **Open Book** Assessment.
4. For Section A, use the OCR form provided (**use 2B pencil**).
You will still need to hand over the entire paper as the MCQ section will not be archived.
5. For Section B, answer **ALL** questions within the **boxed space**.
If you leave the boxed space blank, you will get automatic 1 mark.
However, if you write at least a single character and it is totally wrong, you will get 0 mark.
You can use either pen or pencil. Just make sure that you write **legibly!**
6. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
Read all the questions first! Some questions might be easier than they appear.
7. You can assume that all **logarithms are in base 2**.
8. Please write your Student Number only. Do **not** write your name.

A	0								
---	---	--	--	--	--	--	--	--	--

This portion is for examiner's use only

Section	Maximum Marks	Your Marks
A	39	
B	61	
Total	100	

A Multiple Choice Questions ($13 \times 3 = 39$ marks)

Select the **best unique** answer for each question. Each correct answer worth 3 marks.

THE REST OF PAGE 2 IS REDACTED.

THE ENTIRE PAGE 3 IS REDACTED.

THE ENTIRE PAGE 4 IS REDACTED.

THE ENTIRE PAGE 5 IS REDACTED.

B Essay Questions (61 marks)

B.1 Strategy Making as Antique Collector (30 marks)

You are a collector of antiques. In recent news, there will be a prestigious once-in-a-lifetime bazaar selling rare antiques you do not want to miss.

In that bazaar, there will be n types of antiques being sold. The i -th type of antique has an initial price of p_i dollars and has a quality grade of q_i stars. While you can assume that each type of antique has an “unlimited” stock of antiques, you actually cannot buy many of them. This is because, due to their rarity, if you buy k many antiques of type i , the organizer will charge you $k^2 \cdot p_i$ dollars (so it gets more expensive the more you buy). Also, it is not possible to buy an item fractionally.

Let’s say your budget is only M dollars and your happiness level is defined as the total quality grade stars you can get by buying the antiques. Your task is to find the maximum possible happiness level! (Assume, all p_i, q_i and M are positive integers.) We refer to this problem as the *Rare Antique Collection* problem.

B.1.1 Test your understanding [4 Marks]

Let’s have an example. Suppose that the bazaar only sells $n = 2$ types of antiques. The first type has an initial price of $p_1 = 30$ dollars and a quality grade of $q_1 = 2$ stars. The second type has an initial price of $p_2 = 100$ dollars and a quality grade of $q_2 = 3$ stars.

If your budget is $M = 125$ dollars, then your maximum possible happiness level will be 4 stars. This is achieved by buying 2 antiques for the first type and 0 antiques for the second type, as the total price will be $2^2 \cdot 30 + 0^2 \cdot 100 = 120$ dollars (within the budget) and the happiness level is $2 \cdot 2 + 0 \cdot 3 = 4$ stars.

If your budget is $M = 500$ dollars instead, then what will be your maximum possible happiness level in this scenario (2 marks)? Write down how many antiques you should buy for the first and the second types to achieve that happiness level (also 2 marks)!

B.1.2 How happy you can be? (6 marks)

Suppose Q be the maximum happiness level (stars) any item can give, i.e., $Q = \max_i q_i$. Show that the maximum possible happiness level you can attain is at most $Q\sqrt{nM}$. (You may use the following inequality without proving it: For any two sequences of numbers (a_1, a_2, \dots, a_n) and (b_1, b_2, \dots, b_n) , $(\sum_{i=1}^n a_i b_i)^2 \leq (\sum_{i=1}^n a_i^2) \cdot (\sum_{i=1}^n b_i^2)$.)

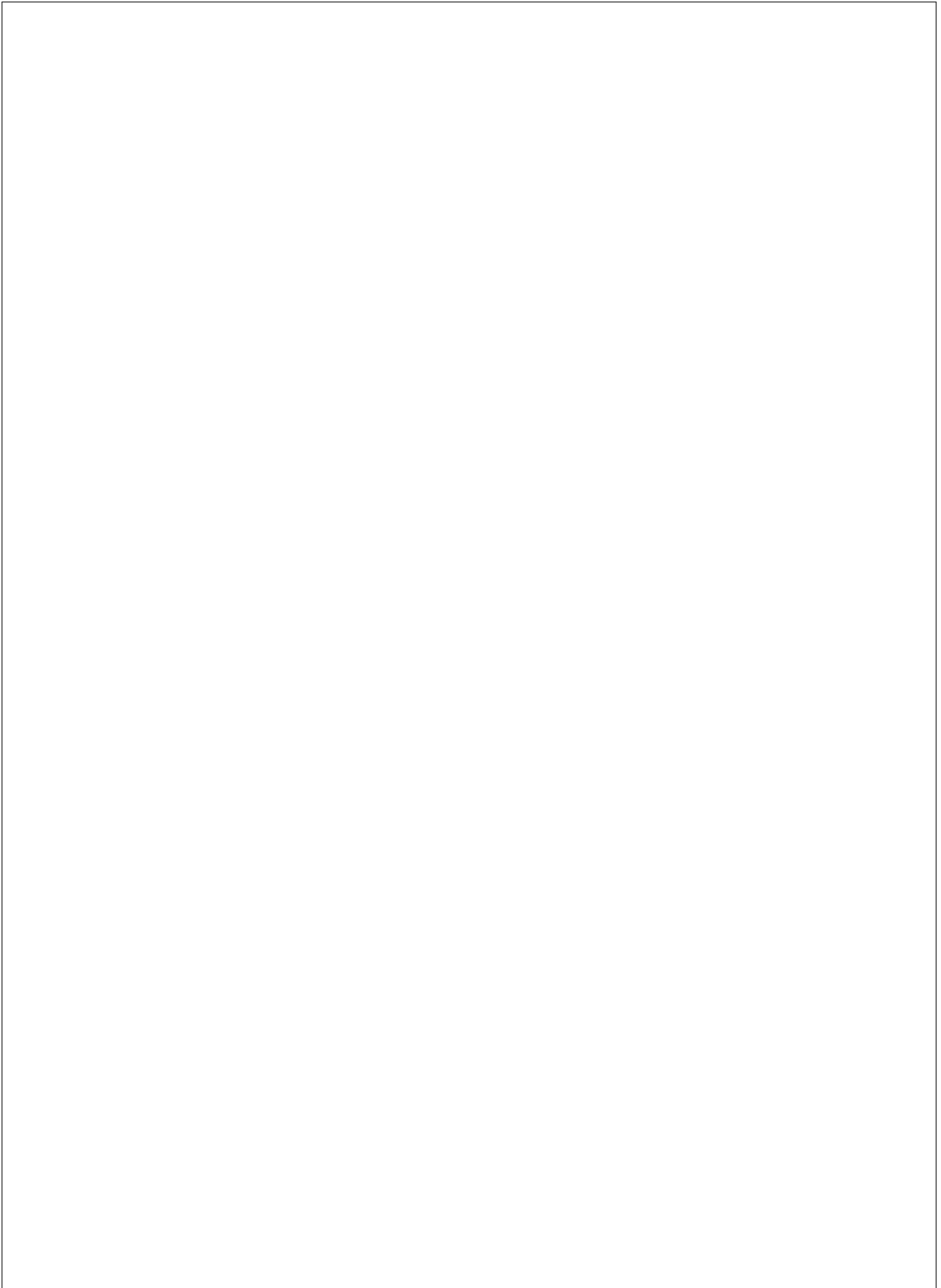
You get a partial marks of 3 if you could only show an upper bound of $Qn\sqrt{M}$ (instead of $Q\sqrt{nM}$).

B.1.3 Design an algorithm for the Rare Antique Collection problem (10 Marks)

Design an algorithm to compute the value of the maximum possible happiness level. To get the full marks, your algorithm should run in $O(Qn^{3/2}M)$ time. You may assume the statement given in Question B.1.2 even if you can't prove it. (**Hints:** Try to find a similarity with the Knapsack problem discussed in the lecture, and try to come up with a Dynamic Programming algorithm.)

Provide a clear description of your algorithm, including the optimal substructure, and a detailed running time analysis.

You get a partial marks of 7 if your algorithm runs in $O(nM^{3/2})$ time, and a partial marks of 3 if your algorithm runs in $O(M^{n/2})$ time.



B.1.4 The rare antique collection problem is NP-complete! (10 Marks)

Prove that the decision version of the Rare Antique Collection problem is NP-complete. (You may show a reduction from any of the NP-complete problems introduced in the lectures/tutorials/assignments/practice sets, including Circuit Satisfiability, CNF-SAT, 3-SAT, Vertex Cover, Independent Set, Max-Clique, Hamiltonian Cycle, Traveling Sales Person, Knapsack, Partition, Subset-sum Problem.)

[**Hints:** You have perhaps already observed some sort of similarity between the Knapsack and the Rare Antique Collection (RAC) problem. So try a reduction from the Knapsack problem. More specifically, for an instance of the Knapsack with n items with value, weight pairs as $(v_1, w_1), \dots, (v_n, w_n)$, and a capacity W , you may try creating an instance of the Rare Antique Collection problem using the following:

- Consider $B = n \cdot \max\{W, v_i, w_i\} + 1$.
- For each item of Knapsack, create an antique type with $p_i = B + w_i$ and $q_i = B + v_i$.
- Create n additional “dummy” antique types each with $p_i = B$ and $q_i = B$.
- Set the budget $M = nB + W$.

]

B.2 Act Greedily in Transporting Hazardous Items (16 marks)

Suppose a pharmaceutical company has a warehouse where several chemical compounds are stored. However, due to a certain space crunch in the current location, the company has decided to relocate the warehouse to a new location slightly far from the current one. So it is required to transport all the chemicals in the warehouse to the new location, and the company has hired a truck for that purpose. Unfortunately, it is impossible to transport certain compounds simultaneously due to security reasons. We call two compounds X and Y *hazard-pair* if they can react with each other and may create an explosion. Thus while transporting chemicals, it is not allowed to put a hazard-pair in the same truck. Now, of course, to reduce the transport cost, the owner of the company wants to have as few trips (by the transport truck) as possible to relocate all the compounds, and thus wants to maximize the number of items transported in each trip (while avoiding transport of a hazard-pair). After knowing that you have learned a lot of algorithms in CS3230, suppose the owner has hired you.

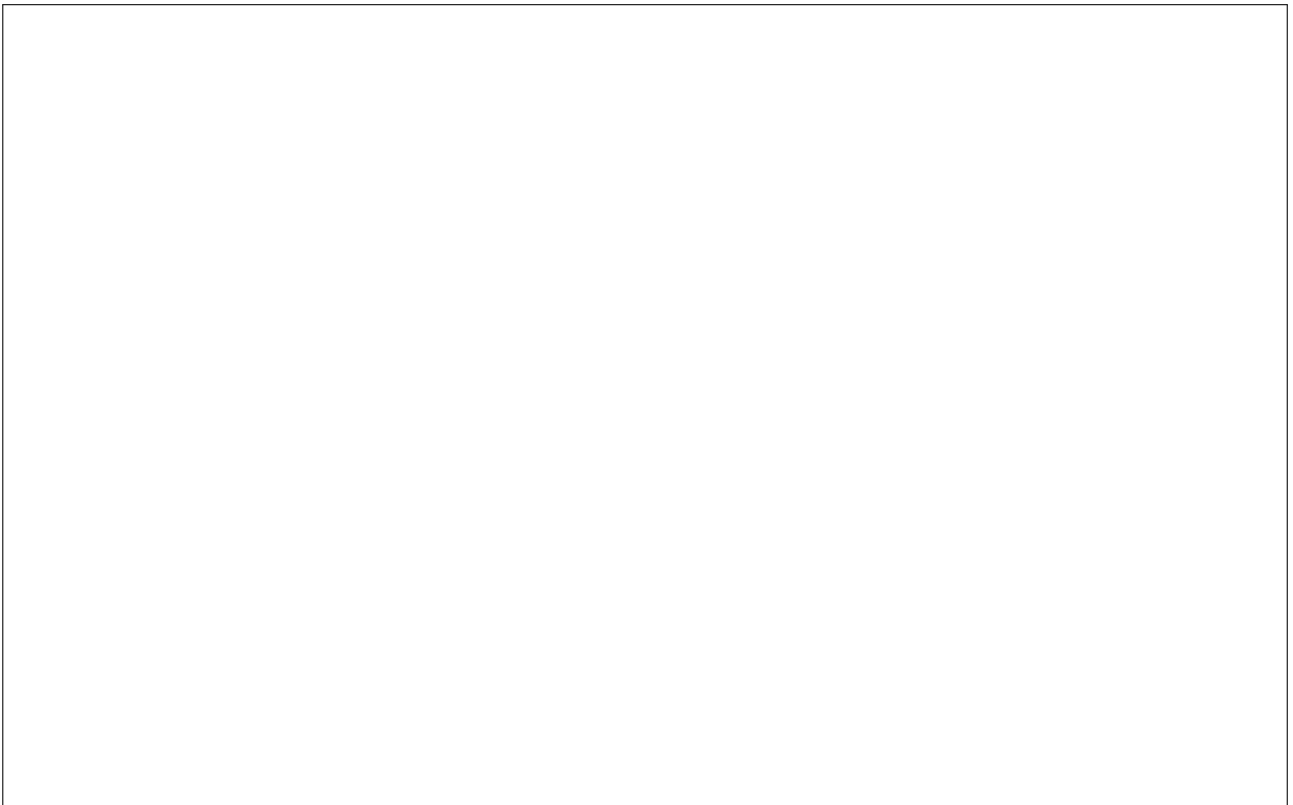
Suppose there are n chemicals present. After seeing the hazard-pairs, you build a *hazard-graph* as follows: The set of vertices is the same as the set of chemicals. Between two vertices, you draw an edge if and only if the corresponding chemicals form a hazard-pair. By carefully looking into the graph, you have observed that no cycle is present in your hazard-graph. Your objective is to design an $O(n)$ time *greedy algorithm* that outputs the maximum number of chemicals so that no hazard-pair is present. Answer the following questions to achieve that.

B.2.1 What is the greedy-choice property (I)? (7 marks)

State a greedy strategy that works for the above problem. Also, provide a formal proof why your greedy strategy leads to an optimal solution to the above problem. (*No marks* will be given without the correct proof.)

**B.2.2 What is the optimal substructure (II)? (4 marks)**

State an optimal substructure property that you would like to use together with your greedy-choice (in B.2.1) to get an optimal solution. Again, provide a formal proof. (*No marks* will be given without the correct proof.)



B.2.3 Design an algorithm (5 marks)

Combine the greedy-choice (in B.2.1) and optimal substructure (in B.2.2) to design an $O(n)$ time algorithm that always outputs an optimal solution. (Provide a pseudocode of your algorithm and a complete running time analysis.)

B.3 Dynamic Table, Stack, and Queue (15 marks)

Consider a standard (FIFO) queue that supports the following operations:

- `PUSH(x)`: Add item x at the end of the queue.
- `PULL()`: Remove and return the first item present in the queue.
- `SIZE()`: Return the number of elements present in the queue.

We can easily implement such a queue using a doubly-linked list so that `PUSH`, `PULL`, and `SIZE` operations take at most c_1 , c_2 , c_3 worst-case time for some constants $c_1, c_2, c_3 > 0$.

Now suppose that you are asked to implement this (FIFO) queue using Python, but you are too lazy to code your own singly-linked list. Instead, you want to just use Python's built-in `list` (which, if you are not aware, implements Dynamic Table (i.e., a resizable array) as discussed in week 7 lecture). Even if you do not know Python, you just need the following information to solve this question:

- You can create an empty Python list L in $O(1)$ by calling `L = []`.
- You can append a new item x to the back of a Python list L in (amortized) $O(1)$ by calling `L.append(x)`. You can assume that in the event Python has to enlarge the internal list/array to two times of the original size because the internal list/array is full, it can do it automatically and we have analyzed this to be amortized $O(1)$ in week 7 lecture.
- You can remove (and return) the last item x from the back of a Python list L in by calling `L.pop()`. Again, you can assume that this operation also runs in (amortized) $O(1)$ even when intermixed with the append operation earlier.
- You can check the length of a Python list L in $O(1)$ by calling `len(L)`.

B.3.1 Python list as Efficient Stack (2 marks)

In pseudo-code (a few short sentences), explain how you can simply use a Python `list` to implement a (LIFO) stack efficiently, i.e., both of these two operations in (amortized) $O(1)$.

- `PUSH(x)`: Add item x at the top of the stack.
- `PULL()`: Remove and return the top item present in the stack.

B.3.2 Two Python lists as Efficient Queue (6 marks)

Now the actual question: In pseudo-code (a few short sentences), explain how you can use *two* Python lists (technically, two stacks, see Section B.3.1) to implement a (FIFO) queue efficiently, i.e., PUSH, PULL, and SIZE operations remains (amortized) $O(1)$. Please reserve this Section for explaining your ideas and only explain the amortized analysis in Section B.3.3. (**Hints:** PUSH(x) to one of the stack, PULL() from the other stack.)

B.3.3 Amortized Analysis (7 marks)

Use **potential method** to show that in any intermixed sequence of PUSH, PULL, and SIZE operations of (FIFO) queue implemented using two Python lists/(LIFO) stacks as answered in Section B.3.2, the amortized cost of each of them is $O(1)$. (Assume, you start with an empty queue).

– END OF PAPER; All the Best –