NATIONAL UNIVERSITY OF SINGAPORE

CS2040S—Data Structures and Algorithms

AY2024/2025, Semester 2

Time Allowed: 2 hours

INSTRUCTIONS TO STUDENTS

1. The assessment paper contains **SIX (6) questions** and comprises **TEN (10) pages** including this cover page.
2. Weightage of each question is given in square brackets. The maximum attainable score is **??**.
3. This is a **CLOSED** book assessment, but you are allowed to bring **ONE** double-sided A4 sheet of notes for this assessment.
4. Please write your Student Number only on the **ANSWER BOOKLET**. Do not write your name. Write all your answers in the space provided in the **ANSWER BOOKLET**.
5. You are allowed to write with pencils, as long as it is legible.
6. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code.

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: MCQs [12 marks]

A. Running time for insertion sort on an array of n strictly decreasing integers takes:

1. $O(1)$ time
2. $O(\log(n))$ time
3. $O(n)$ time
4. $O(n\log(n))$ time
5. $O(n^2)$ time

[2 marks]

For the following statements, please indicate if they are **True** or **False**.

B. If a directed graph has no negative cycles but negative weighted edges, Dijkstra still correctly solves the SSSP problem. [2 marks]

C. Running Dijkstra on a DAG yields a running time of $O(V+E)$. [2 marks]

D. DFS cannot be used to solve the SSSP problem on a undirected, positively weighted, tree. [2 marks]

E. On an undirected, unweighted graph where all edge weights are unique, the minimum spanning tree is unique. [2 marks]

F. An alternative algorithm to create a max-heap (instead of heapify) is to just sort the array in increasing order..

[2 marks]

Question 2: Quickfire [8 marks]

In the following questions, choose **one** algorithm that we should use to solve the problem as efficiently as possible (worst case if deterministic, expected running time if randomized):

A. To find the depth of a rooted tree, we should use: [2 marks]

B. To detect cycles in a directed graph, we should use: [2 marks]

C. To solve the APSP problem where $|E| = |V|$ [2 marks]

D. To decide if two nodes in an undirected, weighted, graph are connected, we should use: [2 marks]

Question 3: Building A Data Structure [10 marks]

A bubble tea shop has a new rewards program that they want their customers to join. We want to build a data structure that tracks the customer with the most amount of points. Here is how the system should work, it should support the following operations:

1. `AdvanceDay()` should update the data structure so that it becomes the next day. Give every customer who has joined and not left one more point.
2. `Join(int customerID, int baselinePoints)` should insert a new customer whose points are just `baselinePoints`. You are promised that customerID values are unique.
3. `MostPoints()` should return the ID of the customer with the most points for the current day, and how many points they have. If there is more than one customer with the highest number of points, return any (only called when the system has at least one customer).
4. `Leave()` should remove the least recently joined customer that still is in the system (only called when the system has at least one customer). (I.e. customers are removed in a first-in-first-out manner)

Note that every day a customer is in the system, they gain a new point for being there.

For example, let's consider this sequence of operations:

- `Join(3, 100)`
- `MostPoints()` $\rightarrow (3, 100)$
- `AdvanceDay()`
- `Join(4, 100)`
- `Join(20, 70)`
- `MostPoints()` $\rightarrow (3, 101)$
- `AdvanceDay()`
- `AdvanceDay()`
- `Leave()`
- `MostPoints()` $\rightarrow (4, 102)$
- `Leave()`
- `MostPoints()` $\rightarrow (20, 72)$

Explanation: On the first day, there is only a single customer, with ID 3, with a baseline of 100 points. On the second day, 2 more customers join. One with ID 4 and baseline of also 100 points, and one with ID 20, with a baseline of 70 points. Since by day 2, customer 3 has accrued one extra point, totalling 101, `MostPoints()` should return $(3, 101)$. After 2 more days, on the fourth day, customer with ID 3 has left, because it was the first to join (therefore the first to leave). Upon doing so, the customer with the most points now is customer 4, having accrued a total of 102 points. After calling `Leave()` again, this time customer 4 leaves, because it was the second to arrive. Then the remaining customer is 20, with a total points of 72.

Your Goal: Build a data structure that supports all of the above operations so that if we used any t of the above operations, it cost $O(t)$ time.

A. Declare any variables or data structures you might need. [2 marks]

B. Describe your algorithm for `AdvanceDay`. [2 marks]

C. Describe your algorithm for `Join`. [2 marks]

D. Describe your algorithm for `MostPoints`. [2 marks]

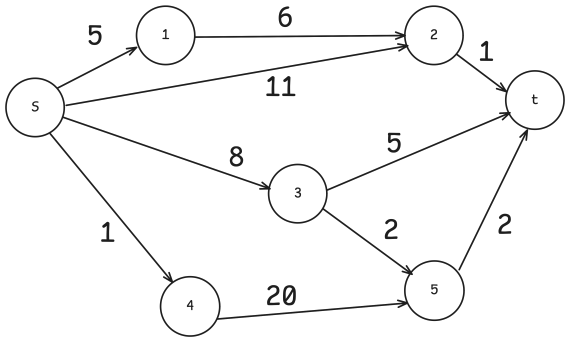
E. Describe your algorithm for `Leave`. [2 marks]

Question 4: Nodle's Cat [10 marks]

Nodle has a new conundrum: He needs to find out how his cat travels between his home and his office. To do this, consider a directed graph $G = (V, E)$ where the edges are positively weighted, and a starting node s, and an ending node t. We know that Nodle's cat will definitely take the shortest path from s to t. The issue is that the shortest path may not be unique. Nodle wants to install a cat detector along **any edge** e that is used in **some** shortest path from s to t.

Your goal: Given a directed, positively weighted graph $G = (V, E)$, and a starting node s, and an ending node t. Describe an algorithm that (as efficiently as possible) outputs the **minimum number of edges that Nodle needs to install cat detectors on**.

For example, consider the following graph:



Then Nodle needs to install detectors along the following edges:

1. $(s, 1)$
2. $(s, 2)$
3. $(1, 2)$
4. $(2, t)$
5. $(s, 3)$
6. $(2, 5)$
7. $(5, t)$

These are all the edges needed in **some** shortest path (of length 12) from node s to node t. Edges like $(3, t)$ and $(4, 5)$ are excluded because it is never path of a shortest path.

A. Give your algorithm that takes as input positively weighted graph $G = (V, E)$, and a starting node s, and an ending node t, and outputs the minimum number of edges needed.

Describe your algorithm that solves the main task. You do not have to re-implement any algorithm covered in class. If you wish to invoke an existing algorithm, mention what it is, and state clearly any modifications you wish to make to those algorithms.

[8 marks]

B. What is the time complexity of your algorithm in Part (A)? If it is deterministic, state the worst case time complexity. If it is randomised, state the expected running time. [2 marks]

Question 5: In this economy? [11 marks]

Considering how the economy is, Nodle has been working for a delivery company in his free time. Today they are trying to solve a new problem: They noticed that customers regularly bought from both Moonbucks and Kallang Fried Chicken at the same time. As such, they want to pair each Moonbucks outlet with the closest Kallang Fried Chicken outlet possible. Formally, you are given an undirected, positively weighted graph $G = (V, E)$, and two sets of nodes M (representing the nodes that are Moonbucks outlets) and K (representing the nodes that are Kallang Fried Chicken outlets):

Your task: Design an algorithm, as efficient as possible, that outputs the distance of closest pair of nodes (m, k) where $m \in M$, and $k \in K$.

A. Describe your algorithm that solves the main task. You do not have to re-implement any algorithm covered in class. If you wish to invoke an existing algorithm, mention what it is, and state clearly any modifications you wish to make to those algorithms. [9 marks]

B. What is the time complexity of your algorithm? If it is deterministic, state the worst case time complexity. If it is randomised, state the expected running time. [2 marks]

Question 6: Last Epoch, anyone? [?? marks]

Nodle's playing an RPG game, and he needs to level his character up. He has $T \geq 0$ time points that he can spend at most, and he has n different quests that he can take. The i^{th} quest q_i is given as a triplet (t_i, e_i, r_i), where it costs t_i time points, will give e_i EXP points, and can only be repeated r_i times at most. Nodle wants to maximise the number of EXP points that he can obtain for his character.

To do so, Nodle can do the i^{th} quest d_i times, where d_i is an integers in the range $0 \leq d_i \leq r_i$, and the total time points incurred as a cost is given as $C = \sum_{i=1}^{n} t_i \cdot d_i$, if he did the i^{th} quest d_i times. The total EXP gained is given as $E = \sum_{i=1}^{n} e_i \cdot d_i$.

Your goal: Give an algorithm that as efficiently as possible, outputs the maximum possible EXP E gained, whilst making sure that $C \leq T$.

For example, let's say there are 3 quests:

1. Quest 1 costs 2 time points, gives 3 XP each, and can be taken up to 4 times
2. Quest 2 costs 3 time points, gives 4 XP each, and can be taken up to 3 times
3. Quest 3 costs 6 time points, gives 8 XP each, and can be taken up to 2 times

then with a time limit of $T = 10$ time points, Nodle can earn 14 EXP by taking Quest 1 twice, and Quest 3 once, costing a total of $2 \times 2 + 1 \times 6 = 10$ time points, and earning a total of $2 \times 3 + 1 \times 8 = 14$ EXP points.

A. Describe your algorithm that solves the main task. You do not have to re-implement any algorithm covered in class. If you wish to invoke an existing algorithm, mention what it is, and state clearly any modifications you wish to make to those algorithms. [8 marks]

B. What is the time complexity of your algorithm? If it is deterministic, state the worst case time complexity. If it is randomised, state the expected running time. [2 marks]

CS2040S—Data Structures and Algorithms

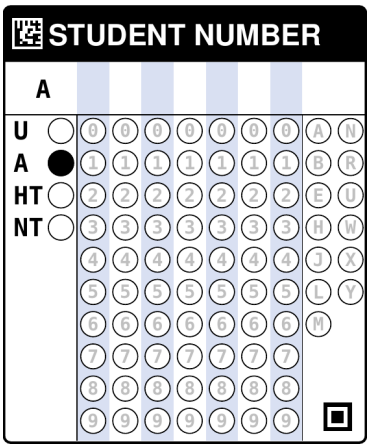
Final Exam Answer Sheet

AY2024/2025, Semester 2

Time allowed: 2 hours

Instructions (please read carefully):

1. Write down your **student number** on the right and using ink or pencil, shade the corresponding circle in the grid for each digit or letter. DO NOT WRITE YOUR NAME!
2. This answer booklet comprises **TEN (10) pages**, including this cover page.
3. All questions must be answered in the space provided; no extra sheets will be accepted as answers. You may use the extra page behind this cover page if you need more space for your answers.
4. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
5. The questions are provided in the question booklet, and if there is any inconsistency in the question, please refer to the question booklet. If there is any inconsistency in the answers, refer to the answer sheet.
6. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
7. Please fill in the bubbles properly. **Marks may be deducted** for unclear answers.



For Examiner's Use Only

Question	Marks
Q1	/ 12
Q2	/ 8
Q3	/ 10
Q4	/ 10
Q5	/ 11
Q6	/ ??
Total	/ ??

Question 1A Running time for insertion sort on an array of n strictly decreasing integers takes: [2 marks]

◯ $O(1)$ time ◯ $O(\log(n))$ time ◯ $O(n)$ time
◯ $O(n\log(n))$ time ◯ $O(n^2)$ time

Question 1B If a directed graph has no negative cycles but negative weighted edges, Dijkstra still correctly solves the SSSP problem. [2 marks]

◯ True ◯ False

Question 1C Running Dijkstra on a DAG yields a running time of $O(V+E)$. [2 marks]

◯ True ◯ False

Question 1D DFS cannot be used to solve the SSSP problem on a undirected, weighted, tree. [2 marks]

◯ True ◯ False

Question 1E On an undirected, unweighted graph where all edge weights are unique, the minimum spanning tree is unique. [2 marks]

◯ True ◯ False

Question 1F An alternative algorithm to create a max-heap (instead of heapify) is to just sort the array in increasing order. [2 marks]

◯ True ◯ False

Question 2A To find the depth of a rooted tree, we should use: [2 marks]

Question 2B To detect cycles in a directed graph, we should use: [2 marks]

Question 2C To solve the APSP problem where $|E| = |V|$ [2 marks]

Question 2D To decide if two nodes in an undirected graph are connected, we should use: [2 marks]

Question 3A Declare any variables or data structures you might need. [2 marks]

Question 3B Describe your algorithm for AdvanceDay [2 marks]

```
void AdvanceDay(){



}
```

Question 3C Describe your algorithm for Join [2 marks]

```
void Join(int customerID, int baselinePoints){

}
```

Question 3D Describe your algorithm for MostPoints [2 marks]

```
void MostPoints(){

}
```

Question 3E Describe your algorithm for Leave [2 marks]

```
void Leave(){









}
```

Question 4A Give your algorithm that takes as input positively weighted graph $G = (V, E)$, and a starting node s, and an ending node t, and outputs the minimum number of edges needed. [8 marks]

```
int count_edges(Graph G, int starting_node s, int target_node t){

}
```

Question 4B

[2 marks]

Question 5A

Your algorithm that solves the main task. [9 marks]

```
int closest_pair(Graph G, List<int> M, List<int> K){




}
```

Question 5B

[2 marks]

Question 6A Your algorithm that solves the main task. [8 marks]

Question 6B

[2 marks]

This page is intentionally left blank.

It may be used as scratch paper.

This page is intentionally left blank.

It may be used as scratch paper.

— END OF ANSWER SHEET —

Question 1A Running time for insertion sort on an array of n strictly decreasing integers takes: [2 marks]

○ $O(1)$ time ○ $O(\log(n))$ time ○ $O(n)$ time
○ $O(n\log(n))$ time ● $O(n^2)$ time

Question 1B If a directed graph has no negative cycles but negative weighted edges, Dijkstra still correctly solves the SSSP problem. [2 marks]

○ True ● False

Question 1C Running Dijkstra on a DAG yields a running time of $O(V+E)$. [2 marks]

○ True ● False

Question 1D DFS cannot be used to solve the SSSP problem on a undirected, weighted, tree. [2 marks]

○ True ● False

Question 1E On an undirected, unweighted graph where all edge weights are unique, the minimum spanning tree is unique. [2 marks]

● True ○ False

Question 1F An alternative algorithm to create a max-heap (instead of heapify) is to just sort the array in increasing order. [2 marks]

○ True ● False

Question 2A To find the depth of a rooted tree, we should use: [2 marks]

BFS or DFS (either option is fine)

Question 2B To detect cycles in a directed graph, we should use: [2 marks]

DFS

Question 2C To solve the APSP problem where $|E| = |V|$ [2 marks]

Dijkstra from every node as a source.

Question 2D To decide if two nodes in an undirected graph are connected, we should use: [2 marks]

BFS or DFS

Question 3A Declare any variables or data structures you might need. [2 marks]

```
An integer to track the current day that it is: current_day;
A max-queue that stores tuples,
supports enqueue, dequeue, and get-max in amortised O(1) time
(from tutorial 7, question 6)

Note here that get-max should return the tuple that
has the maximum second coordinate.
```

Question 3B Describe your algorithm for AdvanceDay [2 marks]

```
            current_day += 1;
```

Question 3C Describe your algorithm for Join [2 marks]

```
            queue.enqueue((customerID, baselinePoints - current_day));
```

Question 3D Describe your algorithm for MostPoints [2 marks]

```
            let max_customer, points = queue.max();
            return (max_customer, points + current_day)
```

Question 3E Describe your algorithm for Leave [2 marks]

```
            queue.dequeue();
```

Question 4A Give your algorithm that takes as input positively weighted graph $G = (V, E)$, and a starting node s, and an ending node t, and outputs the minimum number of edges needed. [8 marks]

```
int count_edges(Graph G, int starting_node s, int target_node t){
    1. Take graph G and reverse the edges (while keeping the
    weight the same), call this reverse_graph;
    // now we want to obtain the distances from s to every node,
    and the distances from every node to t.
    2. Run Dijkstra once on graph G, with node s as the source to
    obtain the distance estimates array dist.
    3. Run Dijkstra once on graph reverse_graph, with node t as the
    source to obtain the distance estimates array reverse_dist.
    // an edge e = (u, v, w) is on a shortest path from s to t if
    // and only if dist(s, u) + w + dist(v, t) = dist(s, t)
    4. let min_dist = dist[t] // which is the shortest distance
        from node s to node t
    5. if dist[u] + weight(u, v) + reverse_dist[v] == min_dist
        then add 1 to the counter
}
```

Question 4B

[2 marks]

$O(ElogE)$ or $O(ElogV)$

Question 5A Your algorithm that solves the main task. [9 marks]

```
int closest_pair(Graph G, List<int> M, List<int> K){
    1. Create a super source node s, that for each node m in M,
        s has an edge to node m with weight 0
    2. Also create a super sink node t, that for each node k in K,
        k has an edge to node t with weight 0
    3. Run Dijkstra on the new graph, with s as the source,
        and return dist(s, t) as the answer
}
```

Question 5B

[2 marks]

$O(E \log V)$ or $O(E \log E)$

Question 6A Your algorithm that solves the main task. [8 marks]

Let $Knapsack(n,T)$ represent the optimal solution for solving the problem for n items, subject to time limit T. Let $r = min(\lfloor \frac{T}{t_n} \rfloor, r_i)$ be the maximum number of copies that we can take of quest n. Then $Knapsack(n,T) = \max_{j=0,1,2...,r} \{Knapsack(n-1, T - j \cdot t_n) + e_n\}$.

```
We will use a max-queue from tutorial 7, question 6 that will store integers.
From now on referring to it as max-queue.

// Note: 0-indexed, so row i corresponds to quest i and so on
Create a DP table with (n + 1) rows and T + 1 columns.
Fill the entire table with 0.
For row_index from 1 to n (inclusive)
    let i = row_index
    For start_col_index from 0 to (t_i - 1) (inclusive)
        Create a new empty max-queue
        For multiple from 0 to (T / t_i)  (inclusive)
            enqueue(Table[row_index - 1][start_col_index + multiple * t_i]
                - multiple * e_i)
                    into max-queue

            if max-queue has more than (r_i + 1) elements
                dequeue an from the max-queue

            let max_value = maximum element of max-queue

            set Table[row_index][start_col_index + multiple * t_i]
                to max_value + (multiple * e_i)
```

Question 6B

[2 marks]

O(nT)