

NATIONAL UNIVERSITY OF SINGAPORE

CS2040 – DATA STRUCTURES AND ALGORITHMS

(Semester 1: AY2022/23)

Answers & Marking Scheme

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Do NOT flip / turn over the test paper until you are told to do so
2. Write your **student number** on pages 1, 3 AND 5 of the answer sheet. Do **NOT** write your name! Clearly **shade** your student number on page 1 too
3. Do NOT rearrange the answer sheet, or add/remove staples. **COMPLETELY shade** the bubble for each answer using a fairly **dark pencil**. You may answer the bonus question using pen or pencil
4. **Submit only the answer sheet** at the end of the assessment. It is your responsibility to ensure that you have submitted it, and submitted the correct answer sheet
5. If you fail to submit the correct answer sheet, fail to provide **correct particulars** or prevent the options from being **automatically detected** by software, we will consider it as if you did not submit your answers. In the best case, **marks will be deducted**
6. No extra time will be given at the end of the assessment for you to write your particulars, shade the answer sheet or write/transfer answers. You must do them **before** the end of the assessment
7. This paper consists of **17** questions. Not more than one option should be shaded per grid. The question paper comprises fourteen (**14**) printed pages including this front page and blank page 9. The answer sheet comprises five (**5**) printed pages, we will disregard the 6th page
8. This is an open-hardcopy-notes assessment but **WITHOUT** electronic materials
9. Marks allocated to each question are indicated. Total marks for the paper is **80**
10. The use of electronic **calculator** is **NOT** allowed

<i>Sect Qn</i>	<i>Max</i>	<i>Marks</i>
S1 Q1-7	32	
S2 Q8-12	15	
S3 Q13	8	
S3 Q14	8	
S3 Q15	10	
S3 Q16	6	
S3 Q17	1	
Total	80	

Legend
Easy
Medium
Difficult

Question 1 – Warmup

L is a `java.util.LinkedList<Integer>`, **T** is a `java.util.TreeSet<Integer>` and **PQ** is a `java.util.PriorityQueue<Integer>`. Each of the 3 data structures contains **N** elements.

For **each of Q1a-g independent of one another**, the code snippet runs correctly, you are to choose the best answer for its time complexity:

<input type="radio"/> $O(\log(\log(N)))$	<input type="radio"/> $O(\log(N))$	<input type="radio"/> $O((\log(N))^2)$	<input type="radio"/> $O(\sqrt{N})$	<input type="radio"/> $O(\sqrt{N}\log(N))$	<input type="radio"/> $O(N)$
<input type="radio"/> $O(N \log(N))$	<input type="radio"/> $O(N^{1.5})$	<input type="radio"/> $O(N^{1.5} \log(N))$	<input type="radio"/> $O(N^2)$	<input type="radio"/> $O(N^2 \log(N))$	<input type="radio"/> $O(N^3)$

Code Snippet Q1a

```
System.out.println(T.ceiling(T.first()));
```

$O(\log(N))$

Code Snippet Q1b

```
PQ.remove(key);
```

$O(\log(N))$

Code Snippet Q1c

```
System.out.println(T.headSet(key).size());
```

`headSet(key)` returns a view of the portion of this set whose elements are strictly less than `key`

$O(N)$

Code Snippet Q1d

```
while (!PQ.isEmpty()) System.out.println(PQ.poll());
```

$O(N \log(N))$

Code Snippet Q1e

```
for (int key : T) // internally uses TreeSet's iterator
    System.out.println(key);
```

$O(N)$

Code Snippet Q1f

```
for (Integer key = T.last(); key != null; key = T.lower(key))
    System.out.println(key);
```

$O(N \log(N))$

Code Snippet Q1g

```
for (int times = 1; times <= L.size(); times++)
    System.out.println(L.get(rand(0, L.size()-1)));
```

`rand(0, L.size()-1)` runs in $O(1)$ time and returns some integer from 0 to `L.size()-1` inclusive.

$O(N^2)$

Now, each of **Q2-7** is **INDEPENDENT** of one another, and **worth more marks** than each part in Q1...

Question 2

In an AVL tree, you are given 2 nodes **A** and **B** which are at the same *level*. The **maximum possible difference** in *heights* of the subtrees rooted at **A** and **B** is:

<input type="radio"/> 0	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4
<input type="radio"/> up to twice the height of the shorter of the 2 subtrees			<input type="radio"/> (could be more than those listed)	

Q3 Ali is F due to “or if the current node is already at A or B” – need to find its parent, not that node itself.

Question 3

You are given a reference to 2 nodes **A** and **B** in a BST. A Node is defined as:

```
class Node {
    int item, height; // height of subtree rooted at this Node
    Node left, right, parent;
}
```

You want to find the lowest possible Node that is an ancestor/parent of BOTH **A** and **B** in $O(h)$ time or better, **h** being the height of the BST. Ali, Balu and Charlie each have a suggestion:

Ali – Start from the root of the BST and navigate down ONE path. At each node, check whether **A** and **B** are in two different subtrees of the current node, or if the current node is already at **A** or **B**. If so, the current node is the answer. Otherwise, recurse in the direction where both nodes are in. **F(1.7)**

Balu – Since we have a reference to both nodes, the answer can be found in $O(1)$ time as it is a function of the height attributes of **A** and **B**. **F(1.0)**

Charlie – Since we have a reference to both nodes, we can store a list of nodes along the path from **A** to root, and another list of nodes along the path from **B** to root. Examining these 2 lists one element at a time from the back will give us the answer. **T(1.3)**

Whose approach(es) work, if any at all?

<input type="radio"/> None	<input type="radio"/> Ali only	<input type="radio"/> Balu only	<input type="radio"/> Charlie only
<input type="radio"/> Ali & Balu only	<input type="radio"/> Ali & Charlie only	<input type="radio"/> Balu & Charlie only	<input type="radio"/> Ali, Balu and Charlie

Question 4

You have N boxes labelled $0..(N-1)$, each painted with some colour. Initially, there are N groups, one box in one group. You can repeatedly combine 2 groups together, one pair at a time, but all boxes in the combined group will be repainted to one colour (could be any colour, new or old). You want to keep track of the colour of each box efficiently.

Ivan says "UFDS is suitable here..."

X – UFDS merges a pair of groups efficiently without having to mutate/shift many boxes. T(1.3)

Y – given a box, UFDS allows efficient finding and tracking of the box's group's properties. T(1.0)

Z – BUT for fairly efficient operations, **BOTH** union-by-rank **AND** path compression must be used, otherwise a sequence of N operations will run $\gg O(N \log N)$ time (much worse than $O(N \log N)$ time)." F(.7)

You don't quite agree with Ivan though... Or do you?

<input type="radio"/> Disagree with all 3	<input type="radio"/> Agree with X only	<input type="radio"/> Agree with Y only	<input type="radio"/> Agree with Z only
<input checked="" type="radio"/> Agree with X, Y only	<input type="radio"/> Agree with X, Z only	<input type="radio"/> Agree with Y, Z only	<input type="radio"/> Agree with X,Y,Z

If BOTH UBR + PC are used, a sequence of N operations will run in practically $O(N)$ time.

If only one is used, a sequence of N operations will not exceed $O(N \log N)$ time.

Question 5

Tom says: "In general,

X – Dijkstra's SSSP algorithm will NOT work correctly on a graph with cycles. F (wrong -> -1)

Y – Prim's MinST algorithm will NOT work correctly on a graph with cycles. F (wrong -> -2)

Z – Bellman Ford's SSSP algorithm will NOT work correctly on a graph with cycles." F (wrong -> -1)

How about you?

<input checked="" type="radio"/> Disagree with all 3	<input type="radio"/> Agree with X only	<input type="radio"/> Agree with Y only	<input type="radio"/> Agree with Z only
<input type="radio"/> Agree with X, Y only	<input type="radio"/> Agree with X, Z only	<input type="radio"/> Agree with Y, Z only	<input type="radio"/> Agree with X,Y,Z

All 3 algos can work with cyclic graphs...

SSSP algorithms may give wrong answer on graphs with -ve cycles, not merely cycles.

MST algorithms are typically used on cyclic graphs, if your undirected connected graph is acyclic you don't even need an MST algorithm, you can just take all $V-1$ edges to form the MST.

Question 6

You want to implement your OWN hash table that can map and store up to **N** keys (**N** is known beforehand and will not change) of type **K**, along with associated values of type **V**. You are able to find a *perfect hash function* $\text{hash}(\text{key})$.

Which of these statement(s), if any, are correct:

X – We can implement the table as a direct addressing table to store the **N** key-value pairs. **F(1.0)**

Y – There is no need for collision resolution, why bother chaining / probing? **T(1.0)**

Z – We can fix the table size at exactly **N** since there will be no need for expansion and rehashing. **F(1.0)**

<input type="radio"/> Disagree with all 3	<input type="radio"/> Agree with X only	<input type="radio"/> Agree with Y only	<input type="radio"/> Agree with Z only
<input type="radio"/> Agree with X, Y only	<input type="radio"/> Agree with X, Z only	<input type="radio"/> Agree with Y, Z only	<input type="radio"/> Agree with X,Y,Z

Perfect hash function means there will be no collisions, but may not be minimal.

Question 7

You want to create your own social network e.g. Facebook, Instagram, ... You have users as vertices and friendships / follows between users as edges. You design this graph **G** for the day when your program is used by billions of users worldwide...

Which of these statement(s), if any, are correct:

X – If we use an adjacency matrix to store **G**, we will likely need MASSIVE amounts of memory that can't fit onto one typical computer. **T(1.4)**

Y – If we use a **hash** map of **tree** maps to store **G** instead of an adjacency matrix, we will likely be better able to balance time vs space requirements. **T(0.8)**

Z – If we use a **hash** map of **hash** maps to store **G**, we should be able to run BFS, DFS, SSSP algorithms fairly efficiently. **T(0.8)**

<input type="radio"/> Disagree with all 3	<input type="radio"/> Agree with X only	<input type="radio"/> Agree with Y only	<input type="radio"/> Agree with Z only
<input type="radio"/> Agree with X, Y only	<input type="radio"/> Agree with X, Z only	<input type="radio"/> Agree with Y, Z only	<input type="radio"/> Agree with X,Y,Z

If 1 byte is needed to store a vertex, exabytes of space (billions of gigabytes) is needed to store the adj matrix. If you use $\text{HM}\langle \text{Vertex}, \text{TM}\langle \text{Vertex}, \text{Edge} \rangle \rangle$ then there will be many less candidate edges to search through when traversing the graph, though the cost of checking incidence is increased from $O(1)$ to $O(\log V)$.

$\text{HM}\langle \text{Vertex}, \text{HM}\langle \text{Vertex}, \text{Edge} \rangle \rangle$ gives the same expected time complexity as using $\text{ArrayList}\langle \text{List}\langle \text{Edge} \rangle \rangle$ for those algorithms, though the run time is slower due to repeated hashing and collision resolution.

Section 2 – Read the scenario carefully and use it to answer Questions 8-12**[15 marks]**

There are **N** cards placed in a line. Each card has a **positive integer number** written on it. Both **N** and the number on a card may be very large. The numbers on each card, in sequence, is given in an integer array **cards**.

You are also given 5 **independent** problems, as well as 10 possible solutions summarized by their **main** algorithm and/or **main** data structure(s) besides those given to you. For each of **Q8-12**, choose:

(a) ONE solution summary that is the **most suitable and most efficient**

<input type="radio"/> S1	<input type="radio"/> S2	<input type="radio"/> S3	<input type="radio"/> S4	<input type="radio"/> S5
<input type="radio"/> S6	<input type="radio"/> S7	<input type="radio"/> S8	<input type="radio"/> S9	<input type="radio"/> S10

and

(b) best answer for the time complexity of the chosen solution summary

<input type="radio"/> $O(\log(\log(N)))$	<input type="radio"/> $O(\log(N))$	<input type="radio"/> $O((\log(N))^2)$	<input type="radio"/> $O(\sqrt{N})$	<input type="radio"/> $O(\sqrt{N}\log(N))$	<input type="radio"/> $O(N)$
<input type="radio"/> $O(N \log(N))$	<input type="radio"/> $O(N^{1.5})$	<input type="radio"/> $O(N^{1.5} \log(N))$	<input type="radio"/> $O(N^2)$	<input type="radio"/> $O(N^2 \log(N))$	<input type="radio"/> $O(N^3)$

Marking Scheme

- Any part (a) or part (b) with two or more options shaded will be treated as being completely wrong
- To discourage spamming of a solution, no **solution** (for part (a)) should be picked more than **once across questions** – If that happens, only the *highest mark* for that solution will be awarded *once* across the entire section
- The best answer for time complexity in part (b) may or may not be repeated across questions
- Each solution (for part (a)) is worth somewhere between 0-3 marks
- The choice of each part (b) will cause the awarded mark from part (a) to be scaled by a factor (i.e. multiplied), between 0.5 (completely wrong) and 1.0 (correct option with respect to *your* chosen ADT in part (a))

Solution Summaries

- S1: No data structure needed, algorithm involves combinatorics
- S2: Linked list problem
- S3: Stack problem
- S4: FIFO Queue, sliding window algorithm
- S5: Hash table problem
- S6: Balanced BST problem
- S7: Dijkstra's SSSP algorithm
- S8: Bellman-Ford SSSP algorithm
- S9: Prim's Minimum Spanning Tree algorithm
- S10: Kruskal's Minimum Spanning Tree algorithm

Question 8

Given `cards`, find and output the number of contiguous card subsequences, with at least 2 cards, that start and end with the same-numbered card.

E.g. if `cards = {1, 2, 1, 3, 1, 2, 1}`, then the answer is 7, because there are 7 such subsequences:
 $\{1, 2, 1, 3, 1, 2, 1\}$, $\{1, 2, 1, 3, 1\}$, $\{1, 3, 1, 2, 1\}$, $\{1, 3, 1\}$, $\{1, 2, 1\}$, $\{2, 1, 3, 1, 2\}$

S5 $O(N)$ – Similar to Macarons but track frequencies of elements instead of that of prefix sums.

Question 9

Given `cards`, find and output the length of the longest contiguous card subsequence that is strictly-increasing.

E.g. if `cards = {1, 4, 4, 1, 1, 1, 3, 5, 7, 9, 2, 3}`, then the answer is 5, which is the length of:
 $\{1, 3, 5, 7, 9\}$

S4 $O(N)$ – Can actually be done using comparison, with no physical DS, but that is not an option.

Question 10

You are given `cards` and a positive integer `power`. You start off at the leftmost card. Each turn, from where you are currently at, you can jump exactly `k` cards left or right where `k` is the number on the card you are at, provided a card exists at that position you are about to jump to.

Each jump from card **A** to card **B** lowers your `power` by $1 + |\text{number on A} - \text{number on B}|$ (i.e. 1 more than the absolute value of the difference in numbers). Your `power` never increases. You have to stop when you do not have enough `power` to make a jump.

Find and output the largest card number that you can possibly land on before you have to stop.

E.g. if `cards = {3, 4, 3, 2, 4, 1, 5}` and `power = 9`, then the answer is 5:
 card with number 3 to card with number 2 lowers `power` by 2 to 7
 card with number 2 to card with number 1 lowers `power` by 2 to 5
 card with number 1 to card with number 5 lowers `power` by 5 to 0

E.g. if `cards = {3, 4, 3, 2, 4, 1, 5}` and `power = 8`, then the answer is 4:
 card with number 3 to card with number 2 lowers `power` by 2 to 6
 card with number 2 to card with number 1 lowers `power` by 2 to 4
 card with number 1 to card with number 4 lowers `power` by 4 to 0
 but unable to jump from card with number 1 to card with number 5 as
`power` needs to be at least 5

S7 $O(N \log(N))$ – Directed graph of (`cards`, possible jumps) with up to 2 edges per vertex. Run Dijkstra's SSSP algo but do stop before distance estimate is greater than `power`. Maintain largest card number of any visited vertex (i.e. when dequeued and not yet exceeding `power`, NOT when relaxing edges).

Question 11 – Looks similar to Q10 but different. Read carefully!

You are given `cards`. You start off at the leftmost card and with a positive integer **power** (which you are to decide on). Each turn, you can jump from where you are at **to any other card**.

Each jump from card **A** to card **B** costs $1 + |\text{number on A} - \text{number on B}|$ (i.e. 1 more than the absolute value of the difference in numbers) on your **power**, but you will **restore your power to the initial value each jump**. You have to stop when you do not have enough **power** to make a jump.

Find and output the smallest **power** you need initially (again, **power** resets to this value after each jump made) such that you will be able to land on all cards in `cards`. It is alright for a card to be visited multiple times.

E.g. if `cards = {3, 4, 3, 2, 4, 1, 5}`, then the answer is 2:

card with number 3 to card with number 3 requires 1 power
 card with number 3 to card with number 4 requires 2 power
 card with number 3 to card with number 2 requires 2 power
 card with number 4 to card with number 4 requires 1 power
 card with number 2 to card with number 1 requires 2 power
 card with number 4 to card with number 5 requires 2 power

S6/S2 $O(N \log(N))$ – This is a MiniMax problem, but modelling as a graph results in many edges. The best edges to jump to will form a tree. First, you should jump from card with number x to other cards with the same number x . Then, only jump between one card with number x to a card with $\text{pred}(x)$ and another card with $\text{succ}(x)$. Therefore, the power needed is $1 + \text{largest gap between any 2 numbers in sorted seq.}$

Question 12

You are given `cards` and also the condition that the entire sequence of cards can be *folded*. A sequence of cards can be **folded** if:

- the sequence is empty
or
- the first card is a different card from the last but has the same number on it, AND the remaining subsequence can be divided into 1 or more contiguous subsequences that can be **folded**

e.g. `{3}`, `{3, 5}`, `{3, 5, 3}` or `{3, 5, 3, 5}` all cannot be folded

e.g. `{3, 3}`, `{3, 4, 4, 3}`, `{3, 4, 5, 5, 4, 3}`, `{3, 4, 4, 5, 5, 3}` or `{3, 4, 7, 7, 4, 5, 5, 3}` can be folded

Find and output, for each element in sequence, the index of the corresponding first/last card that has the same number in each fold. If there are multiple possible answers, just output any valid answer.

E.g. if `cards = {3, 4, 7, 7, 4, 5, 5, 3}`, then the answer is “7 4 3 2 1 6 5 0”, because:

3 at index 0 pairs up with 3 at index 7, enabling that contiguous subsequence to be folded
 4 at index 1 pairs up with 4 at index 4, enabling that contiguous subsequence to be folded
 7 at index 2 pairs up with 7 at index 3, enabling that contiguous subsequence to be folded

...

S3 $O(N)$ – This is just nested bracket/symbol matching with a recursive definition.

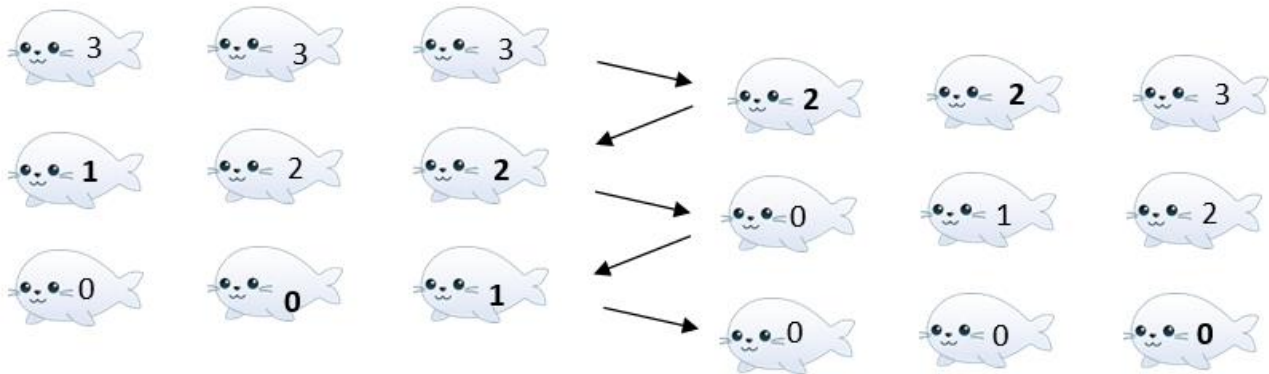
**THIS IS A
BLANK
PAGE**

Section 3 – Read the scenario carefully and use it to answer Questions 13-17

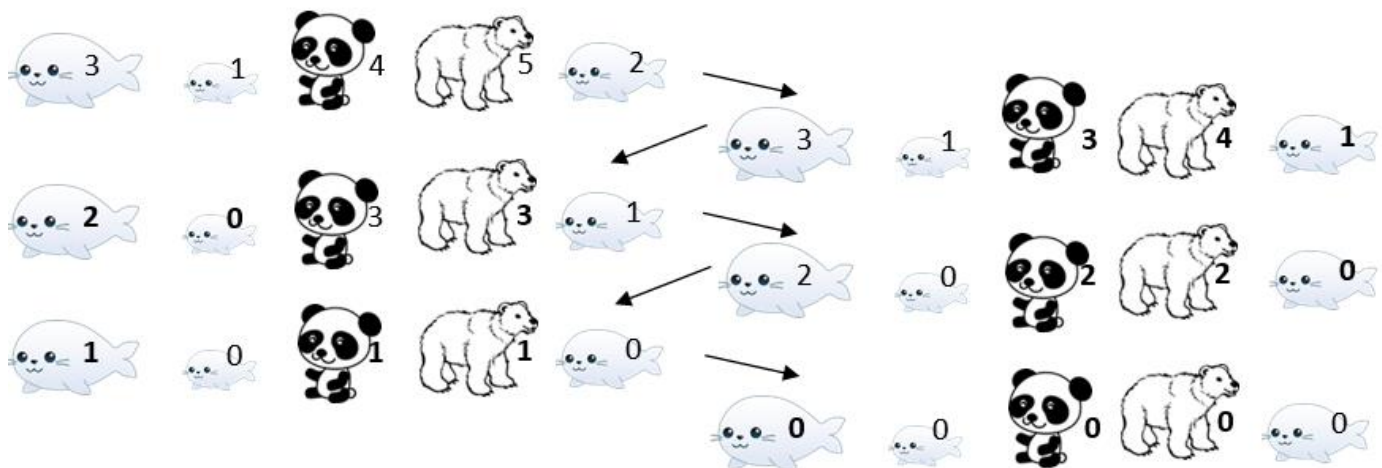
There are N animals¹ in a zoo and they are hungry! Each animal has a positive integer *hunger level* which could possibly be very very large (but never grows). You are tasked to satisfy the hunger of every animal using the *least* number of *feeding cycles*, i.e. all animals should end up with a *hunger level* of 0.

Each *feeding cycle*, you will be given a bucket of k (> 0) fish. Each of the N animals can be fed either 0 or 1 fish, but not more within that same *feeding cycle*. Being fed a fish reduces *hunger level* by 1.

As an **example**, for $\text{hungers} = \{3, 3, 3\}$, $k = 2$, the minimum number of *feeding cycles* required is 5:



Another **example**, for $\text{hungers} = \{3, 1, 4, 5, 2\}$, $k = 3$, the minimum number of *feeding cycles* required is 5:



¹ Images from <http://clipart-library.com>

Question 13 – Understanding the problem**[8 marks == 4 x 2]**

How many *feeding cycles* are minimally required to satisfy the hunger of every animal, for each of:

Q13a. `hungers = {2, 3}`, `k = 5` **Ans: 3**

Q13b. `hungers = {3, 1, 4}`, `k = 3` **Ans: 4**

Q13c. `hungers = {3, 1, 4}`, `k = 2` **Ans: 4**

Q13d. `hungers = {1, 2, 1, 2, 1, 3, 1, 2, 1}` (having length 9), `k = 3` **Ans: 5**

For each question in **Q13a-d**, choose the correct answer:

<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5	<input type="radio"/> 6	<input type="radio"/> 7	<input type="radio"/> 8	<input type="radio"/> 9	<input type="radio"/> 10
-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	--------------------------

Question 14 – Graph Modelling**[8 marks == 4 x 2]**

Ivan suggests that this problem is actually a **shortest path problem**. A **vertex** holds the possible contents of the entire `hungers` array, starting from the initial state before any feeding cycle. An edge represents a *feeding cycle* in which at least 1 fish is fed to an animal, with the destination vertex being the contents of the array after the feed.

What are the characteristics of the graph mentioned above?

Q14a. ☒ **Directed** ☐ Undirected

Q14b. ☐ Weighted ☒ **Unweighted**

Q14c. ☐ Cyclic ☒ **Acyclic**

Q14d. What is the most suitable data structure that should be used to store the graph mentioned above, if any?

<input type="radio"/> Adjacency matrix	<input type="radio"/> Adjacency list	<input type="radio"/> Edge list	<input checked="" type="radio"/> No extra DS, implicit graph
--	--------------------------------------	---------------------------------	---

Question 15 – Shortest Path Algorithm**[10 marks == 5 x 2]**

Assume that there exists a perfect hash function $h(a)$ that can map a vertex (the contents of a possible `hungers` array) as mentioned in **Q14** to an integer hash. If there is a need to keep track of visited vertices, a hash map utilizing the hash function h will be used instead of an array. Assume that the integer hash can be read in $O(1)$ time, no matter how large it is.

Also assume that you **do NOT have control** over the **ordering of the neighbours** of each vertex.

Which of these 4 algorithm(s) correctly solves the problem, ignoring efficiency? The problem may be solved by **AT LEAST 1** of these algorithms. For each of **Q15a-e**, shade the correct option:

<input type="radio"/> Yes	<input type="radio"/> No
---------------------------	--------------------------

Q15a. Run recursive DFS on the vertex containing the initial contents of `hungers`. Stop when a vertex containing all 0s is hit and output/return that number of hops made on that path. **DO** keep track of visited vertices as DFS usually does. **Ans: No** DFS may take a long path, not optimal

Q15b. Run recursive DFS on the vertex containing the initial contents of `hungers`. Stop when a vertex containing all 0s is hit and output/return that number of hops made on that path. Due to the graph's characteristics, we **DON'T** keep track of visited vertices. If there are multiple paths found at a vertex, return the one with the least number of hops. **Ans: Yes** works though inefficient

Q15c. Run BFS on the vertex containing the initial contents of `hungers`. Stop when a vertex containing all 0s is hit and output/return that number of hops made on that path. **DO** keep track of visited vertices as BFS usually does. **Ans: Yes** typical unweighted shortest path algo

Q15d. Run BFS on the vertex containing the initial contents of `hungers`. Stop when a vertex containing all 0s is hit and output/return that number of hops made on that path. Due to the graph's characteristics, we **DON'T** keep track of visited vertices. **Ans: Yes** works though inefficient

Q15e. Just for this part, assume that $k = 2$ to simplify the analysis. Out of the shortest path algorithms you have chosen among **Q15a-d only**, what is the time complexity of the **best** shortest path algorithm?

[Reminder: N is the number of animals and k is the maximum number of animals that can be fed per feeding cycle]

<input type="radio"/> $O(N)$	<input type="radio"/> $O(N^2)$	<input type="radio"/> $O(N^3)$	<input type="radio"/> $O(N^4)$	<input type="radio"/> Other worse polynomial time in N
<input type="radio"/> Exponential in N		<input type="radio"/> Factorial in N		<input type="radio"/> Could be worse than $O(N!)$ even when $k = 2$

Question 16 – Simulation**[6 marks == 3 x 2]**

Remember that k is now not necessarily 2 (back to just $k > 0$), and no longer a graph problem

There are more efficient algorithms to solve this problem. Suppose we **simulate** each *feeding cycle* using different algorithms/data structures to find the answer to the problem, which we will call **A**. In each *feeding cycle*, only the best choice (or one of the many best choices if exist) will be made.

[Reminder: As mentioned earlier, **A** could be very very large]

For each of **Q16a-c** independent of one another, the algorithm correctly solves the problem. Choose the time complexity of using the respective algorithm to find the answer:

<input type="radio"/> $O(N)$	<input type="radio"/> $O(N \log N)$	<input type="radio"/> $O(N^2)$	<input type="radio"/> $O(N^2 \log N)$	<input type="radio"/> $O(AN)$	<input type="radio"/> $O(AN \log N)$
<input type="radio"/> $O(A N^2)$	<input type="radio"/> $O(A N^2 \log N)$	<input type="radio"/> $O(A^2 N)$	<input type="radio"/> $O(A^2 N \log N)$	<input type="radio"/> $O(A^2 N^2)$	<input type="radio"/> $O(A^2 N^2 \log N)$

Q16a.

While at least one animal is still hungry

Sort `hungers` (you decide which algo, choose the best)

For $i = 1..k$, terminating early if out of bounds or a zero is found

Decrement i th-from-the-right element in `hungers`

Output num of while loop passes

$O(AN \log N)$ - In all 3 parts, the outer while loop needs to perform A iterations to complete the simulation

Q16b.

Create **binary heap** (you decide min/max, choose the best) with all hunger levels in `hungers`

Create arraylist of `fedAnimals`, currently empty

While at least one animal is still hungry

For $i = 1..k$, terminating early if heap is empty

Dequeue top element of the heap, decrement, and add to the back of `fedAnimals`

Enqueue all non-zero elements from `fedAnimals` back into the heap

Clear `fedAnimals`

Output number of while loop passes

$O(AN \log N)$

Q16c.

Create a general **binary search tree** with all hunger levels in `hungers`, where each element is a (hunger, frequency) pair

Create arraylist of `fedAnimals`, currently empty

While at least one animal is still hungry

Let $i = k$ and repeat till $i == 0$ or tree is empty

Update and possibly remove some element in the BST

Add a (new hunger, frequency) pair to the back of `fedAnimals`

Add/update all non-zero-hunger elements (pairs) from `fedAnimals` back into the BST

Clear `fedAnimals`

Output number of while loop passes

$O(A N^2)$ – BST may not be balanced \approx

Question 17 – Most Efficient Algorithm**[1 mark (+3 bonus marks?)]**

Remember that k is now not necessarily 2 (back to just $k > 0$)

Legend has it that there is still a more efficient algorithm to solve this problem!

What is the time complexity of the **most efficient** algorithm that can be used to solve this problem?

[Reminder: N is the number of animals. The time complexity should NOT depend on k the maximum number of animals and/or A the answer to the problem (min number of feeding cycles needed)]

<input type="radio"/> $O(\log N)$	<input type="radio"/> $O(\sqrt{N})$	<input checked="" type="radio"/> $O(N)$	<input type="radio"/> $O(N \log N)$
<input type="radio"/> $O(N^{1.5})$	<input type="radio"/> $O(N^2)$	<input type="radio"/> $O(N^2 \log N)$	<input type="radio"/> $O(N^3)$

[Bonus]

Implement the **most efficient** algorithm in **Java** for 3 **bonus** marks:

```
long solve(int[] hungers, int k)
```

[WARNING: Do not waste time on this part, you will receive no credit at all for a solution that is incorrect or that is less efficient than the intended answer]

There are 2 cases. # of feeding cycles could be determined by:

- the hungriest animal
- the ceiling of the ratio of total animal hunger to # possible animals fed per cycle

For the second case, it can be proven that it is always possible to feed the same number of animals in every cycle except the last cycle, i.e. as long as you don't have more fish in the bucket than number of animals, and it is not the last feeding cycle, no fish will be left behind in the bucket.

The only 2 exceptions are when there are more fish in the bucket than number of animals (place cap on effective num of fish), or when there is an animal that is hungrier than that ceiling-of-average (first case).

```
long solve(int[] hungers, int k) {
    k = Math.min(k, hungers.length); // # really fed limited by #animals
    int max = 0;
    long sum = 0L;
    for (int hunger : hungers) {
        max = Math.max(max, hunger);
        sum += hunger;
    }
    long ceilAvg = 1 + (sum - 1) / k;
    return Math.max(max, ceilAvg);
}
```

- End of paper -

Marking Scheme

	A	B	C	D	E	F	G	H	I	J	K	L
Q1a		2										
Q1b		2										
Q1c						2						
Q1d							2					
Q1e						2						
Q1f							2					
Q1g										2		
	A	B	C	D	E	F	G	H				
Q2						1	3					
Q3	1.7	1.0	0.7	3.0		2.3	2.0	1.3				
Q4	0.7	2.0	1.7		3.0	1.3	1.0	2.3				
Q5	3	2	1	2		1						
Q6	2	1	3	1	2		2	1				
Q7		1.4	0.8	0.8	2.2	2.2	1.6	3.0				
	A	B	C	D	E	F	G	H	I	J		
Q13a			2									
Q13b				2								
Q13c				2								
Q13d				1	2							
	A	B	C	D								
Q14a	2											
Q14b		2										
Q14c		2										
Q14d				2								
	A	B	C	D	E	F	G	H				
Q15a		2										
Q15b	2											
Q15c	2											
Q15d	2											
Q15e						1		2				
	A	B	C	D	E	F	G	H	I	J	K	L
Q16a						2						
Q16b						2						
Q16c						1	2					
	A	B	C	D	E	F	G	H				
Q17a			1									

As stated in the question paper:

- if a row has a cell with a score, then all cells in that row that are blank have a score of half of the max score in that row
- each mark will be assigned to one of S1..S10 instead of each question

Q8

	A	B	C	D	E	F	G	H	I	J	K	L
S1												
S2										1.2		
S3										0.2		
S4										0.2		
S5						3.0				2.1		
S6							2.4					
S7												
S8												
S9												
S10												

Q9

	A	B	C	D	E	F	G	H	I	J	K	L
S1						1.2						
S2						2.6						
S3										0.2		
S4						3.0						
S5												
S6												
S7												
S8												
S9												
S10												

Q10

	A	B	C	D	E	F	G	H	I	J	K	L
S1												
S2												
S3												
S4												
S5												
S6												
S7							3.0			2.2	2.1	
S8										2.4		1.7
S9												
S10												

Q11

	A	B	C	D	E	F	G	H	I	J	K	L
S1							1.2					
S2							3.0					
S3												
S4												
S5												
S6							3.0					
S7												
S8												
S9										2.6	2.4	
S10											2.4	

Q12

	A	B	C	D	E	F	G	H	I	J	K	L
S1						1.2						
S2						2.6						
S3						3.0						
S4										0.2		
S5						2.4						
S6							1.6					
S7												
S8												
S9												
S10												