NATIONAL UNIVERSITY OF SINGAPORE

CS2040 – DATA STRUCTURES AND ALGORITHMS

(Semester 1: AY2020/21)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Copy the `ans_blank.txt` text file and rename it to `<your NUSNET ID>.txt` e.g. `e0123456.txt` … Write your student number and NUSNET ID within the appropriate tags of the copied file. Ensure you **answer question 0** as instructed. Failure to do so will prevent your submission from being graded
2. This is an **Open Hardcopy Notes** assessment **WITHOUT** electronic materials. Electronic calculator is **NOT** allowed
3. It is your responsibility to ensure that you have submitted the correct file with the correct particulars and format. If you submit the wrong file, name the file incorrectly, fail to provide correct particulars or change the file contents such that answers cannot be parsed, we will consider it as if you did not submit your answers. In the best case, marks will be deducted
4. No extra time will be given at the end of the assessment for you to write your particulars. You must do it **before** the end of the assessment. However, you may upload your file after the end of the assessment
5. This paper consists of **TWELVE (12)** questions - **Q0** and **8** other inline questions, along with **3** multiline questions - and comprises **ELEVEN (11)** printed pages including this cover page
6. Answer all questions within the text file. Inline answers should be appended to the end of each `#Qx...:` part on the same line. Multiline answers should be written between the appropriate tags with proper indentation and adhering to the line limit. Avoid using the # character in both cases. Do NOT add, modify, remove any tag
7. Marks allocated to each question are indicated. Total marks: **80**

Question 0 **[0 for ENTIRE PAPER if not done satisfactorily!]**

Please read the following NUS Code of Student Conduct (in particular the part on Academic, Professional and Personal Integrity), as well as items B and C below.

(A) **I am aware of, and will abide by the NUS Code of Student Conduct (in particular the part on Academic, Professional and Personal Integrity as shown below) when attempting this assessment.**

- Academic, Professional and Personal Integrity
    1. The University is committed to nurturing an environment conducive for the exchange of ideas, advancement of knowledge and intellectual development. Academic honesty and integrity are essential conditions for the pursuit and acquisition of knowledge, and the University expects each student to maintain and uphold the highest standards of integrity and academic honesty at all times.
    2. The University takes a strict view of cheating in any form, deceptive fabrication, plagiarism and violation of intellectual property and copyright laws. Any student who is found to have engaged in such misconduct will be subject to disciplinary action by the University.
    3. It is important to note that all students share the responsibility of protecting the academic standards and reputation of the University. This responsibility can extend beyond each student's own conduct, and can include reporting incidents of suspected academic dishonesty through the appropriate channels. Students who have reasonable grounds to suspect academic dishonesty should raise their concerns directly to the relevant Head of Department, Dean of Faculty, Registrar, Vice Provost or Provost.

(B) **I have read and understood the rules of the assessments as stated below.**

1. Students should attempt the assessments on their own. There should be no discussions or communications, via face to face or communication devices, with any other person during the assessment.
2. Students should not reproduce any assessment materials, e.g. by photography, videography, screenshots, or copying down of questions, etc.

(C) **I understand that by breaching any of the rules above, I would have committed offences under clause 3(l) of the NUS Statute 6, Discipline with Respect to Students which is punishable with disciplinary action under clause 10 or clause 11 of the said statute.**

3) Any student who is alleged to have committed or attempted to commit, or caused or attempted to cause any other person to commit any of the following offences, may be subject to disciplinary proceedings:

l) plagiarism, giving or receiving unauthorised assistance in academic work, or other forms of academic dishonesty.

To answer Q0, type either your student number or NUSNET ID to declare that you have **read and will abide** by the NUS Code of Student Conduct (in particular, (a) Academic, Professional and Personal Integrity), (b) and (c).

Ans:_____

Section A: Short Answer Qns **[56 marks, 8 x 7]**

For each of the 8 independent questions, identify the:

- tightest big-O asymptotic upper bound for your given algorithm - without justification
- **BEST** data structure(s) **besides** those already given or created by algorithm - **max 50 characters**
- **BEST** algorithm(s) - **max 50 characters**

The 50 characters exclude spaces and punctuation where used in normal English grammar

Use these mnemonics to help you gain more space:

Data Structures

AD	adjacency list	BT	binary tree	PQ	priority queue, max/min?
AL	ArrayList	EL	edge list, of graph	QUE	FIFO Queue
AM	adjacency matrix	HS	HashSet	STK	Stack
AR	Array	HM	HashMap	TS	TreeSet
BBST	balanced binary search tree	LL	LinkedList, BLL/TLL/DLL/CLL?	TM	TreeMap
BST	binary search tree				

Algorithms

BF	breadth first	HSO	heap sort	POT	post-order traversal
BS	binary search	IOT	in-order traversal	PRT	pre-order traversal
BSO	bubble sort (improved)	ISO	insertion sort	QSO	quick sort
BUSO	bucket sort	LOT	level-order traversal	RQSO	randomized pivot quick sort
CSO	counting sort	LS	linear search	RSO	LSD radix sort
DF	depth first	MSO	merge sort	SSO	selection sort
DI	dijkstra's SSSP algo	PM	Prim's MST algo	TSO	topological order / sort

Example Question

Given integer heights, in cm, of **N** real-life people where **N** is very large and odd, find the median height

```
Time complexity : N
                  12 34 5 6789012 345678901234567890123456789012345678901234567890
Best DS         : AR of N heights unordered                    strict 50 char limit -->
Best algo       : CSO, LS, stop at N/2th freq, return index    strict 50 char limit -->
                  123  45  6789 01 23456 7890, 123456 78901234567890123456789012345678
```

Notes

- If there are multiple solutions, choose one instead of listing two or more in your answer
- If the vanilla (plain, as discussed in lectures) version of the algorithm in your answer creates some data structure(s) for its internal use, you do not need to waste space identifying those data structure(s)
- Your solution reads input from the standard input stream, unless otherwise stated

Merely stating the data structure + algorithm (e.g. `N AR CSO`**) will NOT earn you full marks**

Marking will be done by question, and not by each of the 3 fields

Question 1

Given a positive integer **N** where **N** is very large, find the value of 1 + 2 + 3 + 4 + ... + **N**

Question 2

You are given integer lengths, in mm, of **N** undersea cables $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$, ... $\mathbf{d}_\mathbf{N}$ where **N** is very large, as well as a positive integer **K**. Many of the cables are very long and have the exact same length. Find the number of distinct cable lengths for which at least **K** out of the **N** given cables are of that length

Example:

If **K** = 2
and cable lengths are 9876543 9876540 9000000 9876540 9612345 9876540 9000000 9876543
then the answer is 3

Question 3

A temporary one-lane bridge is going to be built to allow **N** heavy vehicles, where **N** is very large, to move across a river in a sequence which cannot be changed. The bridge has a weight limit of **L**, so you need to decide on a value of **K**, the maximum number of vehicles allowed on the bridge at any one time. Once **K** is set, it will not be changed

You are given **N**, as well as floating point values **L**, $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$, ... $\mathbf{w}_\mathbf{N}$ where $\mathbf{w}_\mathbf{i}$ is the weight of the $\mathbf{i}^{th}$ vehicle. Find the highest value of **K** so that no vehicle is at risk of ending up swimming due to the bridge collapsing from excess weight

Question 4

COVID19 pandemic has segmented NUS into 5 zones. Consider instead, an area with **N** people and **Z** zones, where **N** is very large. Suppose each pair of people (**a**, **b**) has a transmission risk $\mathbf{r}_{\mathbf{a,b}}$ (== $\mathbf{r}_{\mathbf{b,a}}$) of spreading some disease. As long as 2 people meet with one another, they incur this risk. Otherwise, they incur 0 risk. For simplicity, assume that people cannot meet in groups, and that their risk remains constant no matter how many times they meet.

The "connected zoned least risk" scenario is one in which, in order of highest priority to lowest:

1. every pair of people (**c**, **d**) are "somehow connected". This means that either:
   - **c** and **d** meet, or
   - **c** and **d** are both "somehow connected" to some other person
2. the number of pairs of people in which the two are from different zones is kept to a minimum
3. total risk (sum of all risks among pairs of people who meet) is kept to a minimum

Your inputs to this problem are the integers **N** and **Z**, an **array** of $\mathbf{z}_1$, $\mathbf{z}_2$, $\mathbf{z}_3$, ... $\mathbf{z}_\mathbf{N}$ which are the zone IDs of each person in sequence, as well as an **adjacency list** of an undirected *complete* weighted graph - ArrayList of LinkedList, where in element **a** of the ArrayList, each node contains the ID of person **b** and risk value $r_{\mathbf{a,b}}$ (besides the reference of the next node). Find the total risk in this "connected zoned least risk" scenario

Scenario for Question 5 to 7

There is a social network of **U** users. **T** of these users, which we will call advertisers, took photographs of an event (just one event for this entire scenario) in a post each and shared the post, at different times, to at least one other user. A user who views a shared post may also share the event with other user(s). Each share has a positive integer rating **r** to determine how relevant the shared post is to the viewing user.

To elaborate, a user first receives and reads ALL the posts shared with him about the event, then either chooses to NOT share posts at all, or to share ALL the posts that have been shared with him to other user(s) at once (this counts as only **ONE** shared post). All **U** users (including an advertiser who receives a shared post) will NEVER share post(s) a second time, and will NEVER receive post(s) about the event after sharing

There are a total of **S** shares. **U**, **T** and **S** are very large. You are given as input:

- the integers **U**, **T** , **S**
- an edge list of **S** edges ($\mathbf{f_1}$ $\mathbf{t_1}$ $\mathbf{r_1}$), ($\mathbf{f_2}$ $\mathbf{t_2}$ $\mathbf{r_2}$), ... ($\mathbf{f_S}$ $\mathbf{t_S}$ $\mathbf{r_S}$) with an edge representing

  a share made by user ID $\mathbf{f_i}$ to user ID $\mathbf{t_i}$ with rating $\mathbf{r_i}$

  as well as
- $\mathbf{a_1}$, $\mathbf{a_2}$, $\mathbf{a_3}$, ... $\mathbf{a_T}$ representing the user ID of the **T** advertisers

Question 5

Often, one or more of these **T** advertisers themselves eventually gets shared a post of the event which originates from another advertiser. Find out the minimum number of shares that the post needed to take, among all shared posts, for this scenario to occur (in which some advertiser receives the shared post)

Question 6

You are also given a user ID **x**. Find the number of advertisers whose post has eventually made its way to user **x**

Question 7

You are also given an advertiser ID **a**. The strength of the relationship between advertiser **a** and some other user **b** is given by the minimum of scores of different paths **a**'s post can take on its way to **b** (If you recall, posts only originate from advertisers). The score along a path is the **product** of all $\mathbf{r_y}$ where **y** is a user who receives the shared post along that path. Find the user with the weakest relationship from advertiser **a**, among users who have received **a**'s post

Question 8

Your input to this problem is an **array** of **N** elements where **N** is very large. Computer memory is very limited, yet you need to sort these **N** elements 'very quickly', and you need to ensure that no matter what elements the array has, they will **always** be sorted 'very quickly'

You learnt that there is a data structure that can be built within the given array itself, and this data structure can help you in sorting 'very quickly' given the above constraints. However, you hear that this sort performs a little slow empirically (under experiments where execution is timed), compared to its competitors. Let's call this the '*fast-but-not-that-fast*' sort

Find a way to sort the elements in the array (still subject to the requirements / constraints in the first paragraph) such that most of the time, it is quicker than that '*fast-but-not-that-fast*' sort mentioned in the second paragraph. In the occasional worst case, you don't mind accepting up to 3 times the time taken by the '*fast-but-not-that-fast*' sort

INLINE answers:

```
#Q1 time:_____________________________
###      12345678901234567890123456789012345678901234567890 ###
#Q1   ds:_____________________________________________________
#Q1 algo:_____________________________________________________
###      12345678901234567890123456789012345678901234567890 ###
```

(Same format for Q2...Q8)...

Section B: Coding Qns **[24 marks == 9 + 8 + 7]**

Question 9 **(9 marks)**

Given an AVL tree of **N** integers where **N** = (2^k - 1) for some integer **k**, **efficiently** create a BST of these same **N** integers, in which all simple paths from any leaf to the root have the same length. **Return the root node** of the **BST**

You may assume that the AVL and BST nodes are instances of the same `Node` class. You have the freedom to either use the same node objects, or create new ones (new objects, NOT a new class)

```
class Node {
      public int item;
      public Node left, right;
      public Node(int i) { this(i, null, null); }
      public Node(int i, Node l, Node r) { item = i; left = l; right = r; }
      // can use getters/setters, or access variables directly without penalty
}
```

Use **Java code**. You may create other helper methods neatly, below this given method, if they help you
[Line limit: **35**]

```
Node morphAVLtoBSTuniformHeight(Node avlRoot) {









}
```

Question 10 **(8 marks)**

You are given an array-based 0-indexed 'max-heap' containing **N** distinct elements, **N** > 1. Just as a bug destroyed the integrity of your linked list in the midterms, another bug here caused exactly one pair of *different* elements (**t**, **b**) within the 'heap' to be swapped, with **b** being the descendant or child of **t**. Sadly, you don't know which pair

In O(**N**) time or better, repair the heap. You can only perform **ONE swap** to undo the damage (Otherwise, without this restriction, you can just rebuild the entire heap, right?)

Hint: How many heap property violations will there be?

Use **Java code**. You may create other helper methods neatly, below this repair method, if they help you

```
void swap(int[] arr, int i, int j) {
      int newI = arr[j]; arr[j] = arr[i]; arr[i] = newI;
}

int getParent(int childIdx) { return (childIdx - 1) / 2; }
```

[Line limit: **35**]

```
void repairMaxHeap(int[] maxHeap) {









}
```

[Difficult] Question 11 **(7 marks)**

You are examining a "world" where all **P** pieces of land (**P** is a large integer) are flat and horizontal, but raised vertically. Each piece of land is at a different height from its immediate neighbours (otherwise, they would be considered the same piece of land)

On each piece of land that is higher than all its neighbours, there is 1 tribe (clan / nation / family) of people. Over time, each tribe establishes itself and sends settlers down to ALL lower adjacent pieces of land (but never upwards), forming 1 new tribe at each piece of land. This occurs only ONCE per tribe. The new tribes eventually get established, sends settlers down to all lower adjacent pieces of land, forming 1 new tribe at each piece... repeating the process

Assuming that the tribes never dissolve / migrate / assimilate / become extinct, complete timeTravel(), which finds the **final number of tribes** in **each piece of land** and stores the answer as the respective element in an array **R**

```
void timeTravel(ArrayList<LinkedList<Integer>> adjList, // L
      long[] numTribesAt) // R
```

You are given:

- an adjacency list **L** of a **directed connected unweighted** graph of adjacent pieces a settler can reach
- a long array **R** of **P** zeroes, for you to set $\mathbf{r_i}$ with the *result of your computation* at piece of land with index **i**

There are **E** pairs of adjacent pieces (**P** - 1) ≤ **E** ≤ 10**P**

As an **example**, with explanation on the next page:

R

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

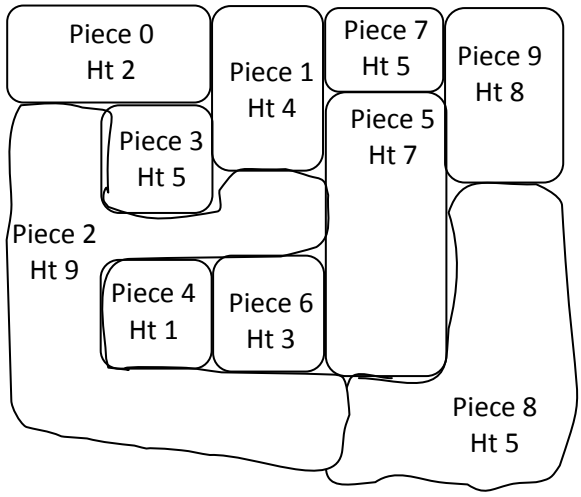
L

```
0 :
1 : 0
2 : 0->1->3->4->5->6->8
3 : 0->1
4 :
5 : 1->6->7->8
6 : 4
7 : 1
8 :
9 : 5->7->8
```

After timeTravel() has completed execution:

R'

0	1	2	3	4	5	6	7	8	9
9	7	1	1	4	2	3	3	4	1

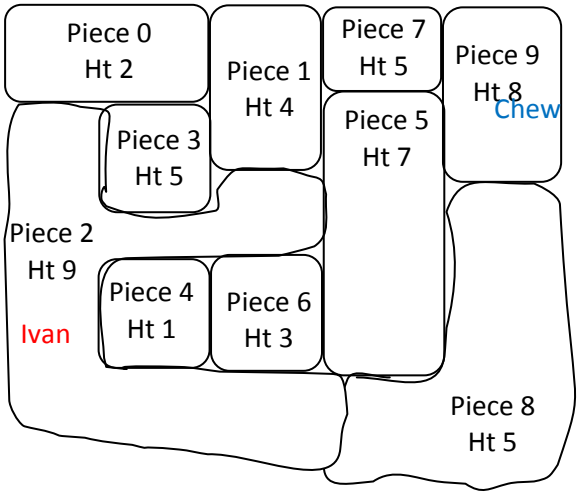


You do not need to worry too much about efficiency (but infinite loop or infinite recursion is NOT ok though). You will get up to **4 bonus marks** if you do this efficiently

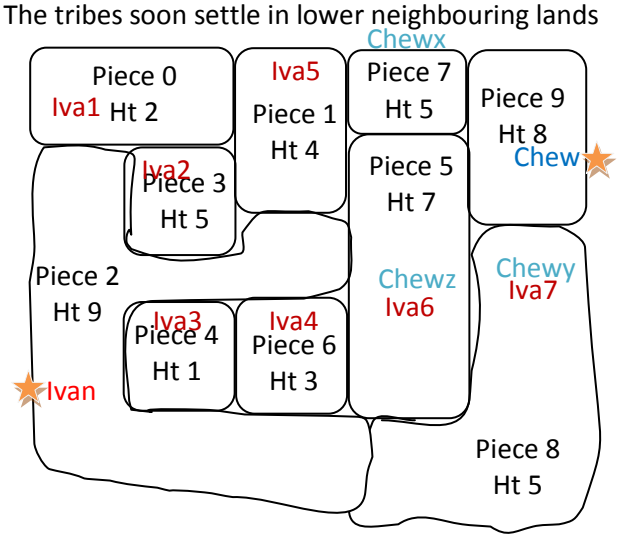
Use **Java code**. You may create other helper methods neatly, below this given method, if they help you. In the event that you can't completely solve the entire problem in time, you may omit (for this question 11 only) the implementation of *methods of algorithms as taught in this module* and just make the appropriate *calls with the right parameters*, for 1m penalty

[You can skip the illustration on this page if you already fully understand the question from the problem statement]

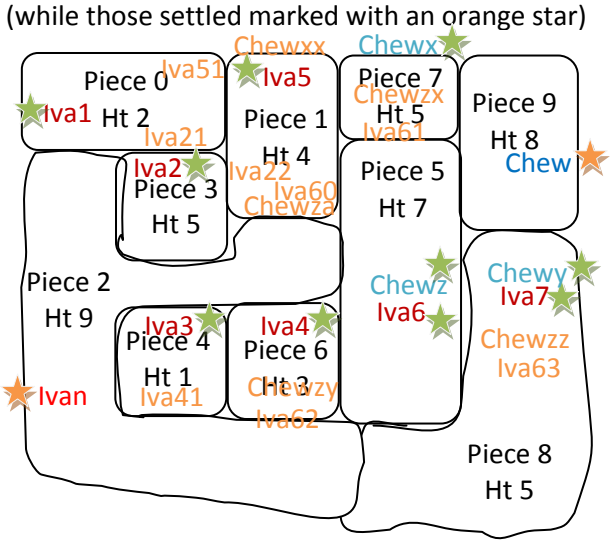
Initially, there are 2 tribes, the tribe of Ivan and the tribe of Chew because each of these lands is higher than all its neighbours



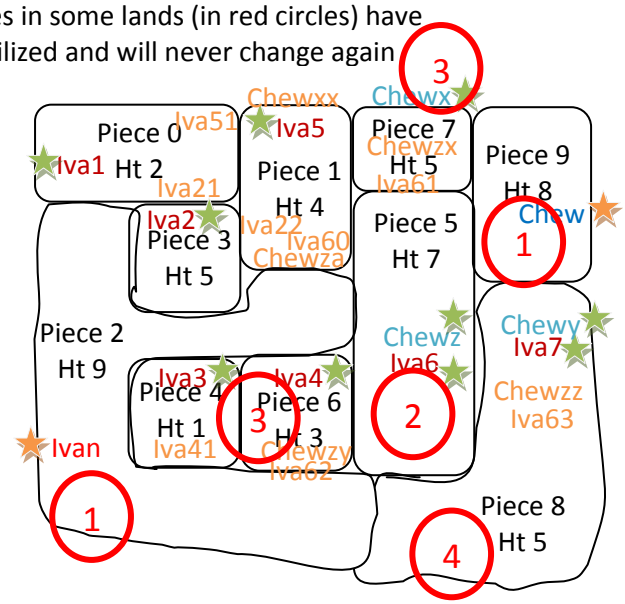
The tribes soon settle in lower neighbouring lands



Later on, tribes who have not settled elsewhere (green star) spread to lower neighbouring lands (while those settled marked with an orange star)

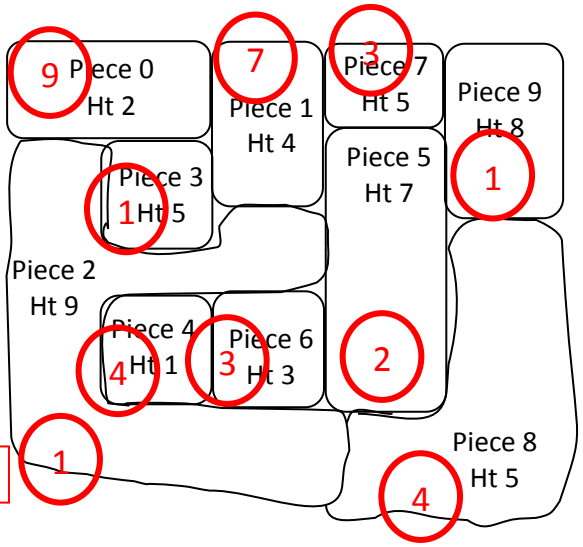


Still at the same point in time, the number of tribes in some lands (in red circles) have stabilized and will never change again



The process repeats itself. Eventually, the number of tribes in each land stabilize...

Each of these number should end up in **R** in sequence



R										
	9	7	1	1	4	2	3	3	4	1

[Line limit: **65**]

```
void timeTravel(ArrayList<LinkedList<Integer>> adjList, // L
      long[] numTribesAt) { // R









}
```

- **End of Paper** -