NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR

Semester 1, AY2020/21

CS2040C – Data Structures and Algorithms

24 November 2020 Time allowed: 2 hours

STUDENT **NO. :** | | | | | | | | | |

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 examination but **WITHOUT** electronic materials
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 contents of the file such that it 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 test for you to write your particulars. You must do it **before** the end of the test. However, you may upload your file after the end of the test
5. This paper consists of **2** inline questions including Q0, and **5** multiline questions. It comprises **eight (8)** printed pages including this front 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 for the paper is **100**
8. The use of electronic **calculator** is **NOT** allowed

Question	***Max***	***Marks***
Q0	-100 if empty	
Q1ab	10	
Q2ab	10	
Q3abcdefg	24	
Q4	11	
Q5ab	10	
Q6	15	
Q7	20	
Total	**100**	

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:

Question 1 **[10 marks, 2 x 5]**

a) Where is/are the possible location(s)/indic(es) of the **2nd most recently added integer** in a **Queue** that currently contains **n** (**n** ≥ 2) distinct integers? Note that if we index the current integers in the Queue, then index 0/**n**-1 is the head/tail of the Queue, respectively.

Ans:

b) Where is/are the **hardest accessible** location(s)/indic(es) of a **Doubly Linked List** that currently contains **n** (**n** ≥ 100 000) distinct integers? Note that index 0/**n**-1 is the head/tail of the Doubly Linked List, respectively. The Doubly Linked List has both head and tail pointers.

Ans:

Question 2 **[10 marks, 2 x 5]**

a) Where is/are the possible location(s)/indic(es) of the **3rd largest integer** in a **Binary Max Heap** of **n** (**n** ≥ 7) distinct integers? Note that we skip index 0 so the root is at index 1.

Ans:

b) Where is/are the possible location(s)/indic(es) of the **2nd smallest integer** in a **Binary Max Heap** of **n** = 14 distinct integers? Note that we skip index 0 so the root is at index 1.

Ans:

Question 3 **[24 marks, 4 x 3 + 3 x 4 = 12 + 12 = 24]**

In the first part, you will be asked to give the best way to implement the following ADTs as discussed in class. Each of the first four questions worth 3 marks.

We give one DEMO question and expected answer that should be followed for the other parts in this question.

DEMO question: The best data structure for list ADT that can support get(i) in O(1), plus insert(i, new-v), search(v), remove(i) in O(n) is:

Ans for DEMO question: *A vector (resizeable-array) as we need get(i) to run in O(1); SLL/DLL cannot do this in O(1); Inserting new-v at index i is O(n) (need to make room at index i), searching for v is also O(n) (may have to search all), removing index i is O(n) (need to close gap after ith element is removed)*

a) The best data structure for stack ADT that supports push(new-v), top(), pop() all in O(1) is:

[Line limit **3**] Ans:

SHORT
AT MOST THREE LINES
ANSWER

b) The best data structure for queue ADT that supports enqueue(new-v), front(), dequeue() all in O(1) is:

[Line limit **3**] Ans:

SHORT
AT MOST THREE LINES
ANSWER

c) The best data structure for priority-queue ADT that can do get_max() in O(1), plus enqueue(new-v) + extract_max() both in O(log n) is:

[Line limit **4**] Ans:

```
SHORT
AT MOST
FOUR LINES
ANSWER
```

d) The best data structure for unordered table ADT that does *not* allow duplicates and support insert(new-v), is_exist(v), remove(v) all in expected O(1) is:

Special note: We roughly know that there are at most **n** keys inside the table ADT at all times.

[Line limit **4**] Ans:

```
SHORT
AT MOST
FOUR LINES
ANSWER
```

In the second part, you will be asked about unnatural (but not necessarily better) ways to implementation the CS2040C related ADTs. Each of the last three questions worth 4 marks.

e) Show how to use C++ std::set or std::map (basically a self-balancing BST like AVL Tree discussed in class) to implement a List ADT of **n** strings. There are four operations to be supported: get(i), insert(i, new-v), search(v), remove(i)).

Special constraints: now get(i) can be **O(log n)** and the other operations can be **O(n log n)**.

[Line limit **5**] Ans:

```
SHORT
AT MOST
FIVE
LINES
ANSWER
```

f) Show how to use C++ std::set or std::map (basically a self-balancing BST like AVL Tree) to implement a Stack ADT of **n** not necessarily distinct integers. There are three operations to be supported: push(new-v), top(), pop().

Special constraints: the three operations can now be **O(log n)** instead of O(1) and there are **at most 1 Billion** push and pop operations in our application.

[Line limit **5**] Ans:

```
SHORT
AT MOST
FIVE
LINES
ANSWER
```

g) Show how to use C++ std::unordered_set or std::unordered_map (basically a hash table, most likely with Separate Chaining Collision Resolution) to implement a Priority Queue ADT of **n** integers. There are three operations to be supported: enqueue(new-v), get_max(), extract_max(), and they have to run faster than O(log **n**)…

Special constraints: the new-v (the keys) are **small integers between [0..100]**, but not necessarily distinct.

[Line limit **5**] Ans:

```
SHORT
AT MOST
FIVE
LINES
ANSWER
```

Question 4 **[11 marks]**

Steven wants to know how many day-type [Monday/Tuesday/..../Sunday] are there in between a given DD1/MM1/YYYY1 to another DD2/MM2/YYYY2, inclusive of both dates.

To make this problem a bit simpler, you are told that **YYYY2 > YYYY1**, YYYY2-YYYY1 is **not more than 10 years**, and **YYYY1 >= 1900** (so you don't have to worry about Gregorian Calendar history). You are also told that the starting date DD1/MM1/YYYY1 **is of a certain day-type**: 0/1/…/6 (to represent Monday/Tuesday/…/Sunday, respectively). You can also assume that **there is a helper function** called `bool isLeapYear(int YYYY)` that will return `true` if YYYY is a leap year (thus 29/02/YYYY exists), like this year 2020, or `false` otherwise.

Example 1: Between 09/08/2020 (it was a Sunday) until 15/11/2020 (it was also a Sunday),
if we are asked to count Sundays, then there were 15 Sundays (remember that there was a Recess week).

Example 2: Between 10/08/2020 (it was a Monday) until 14/11/2020 (it was a Saturday),
if we are also asked to count Sundays, then there were 13 Sundays.

Example 3: Between 28/02/2020 (it was a Friday) until 01/03/2020 (it was a Sunday; notice that 2020 is a leap year),
if we are asked to count Saturday(s), then there was just 1 Saturday as 29/02/2020 was a Saturday.

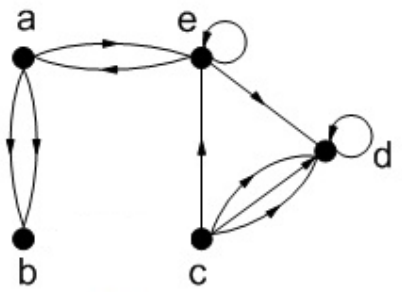
Because the range of years is "small", design **any correct algorithm with its associated data structure** to solve this problem and **analyze its time complexity**!

[Line limit **10**] Ans:

```
MULTILINE
ANSWER
```

Question 5 **[10 marks, 2 x 5]**

a) Initially, you are given the following unweighted directed multigraph below. However, you are told that **all self-loops in this multigraph**, e.g., e→e, **are useless and don't have to be stored**; and the multiple unweighted directed edges between the same pair of vertices (u→v) **can be grouped together into a weighted directed edge** where the weight is the frequency of the multiple edges from u to v, e.g., weight of c→d below is 3.

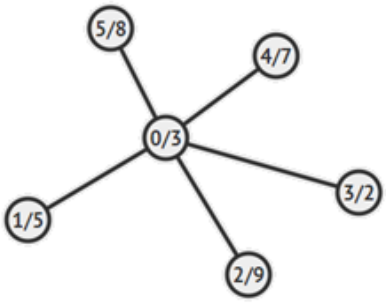


For this sub-question, show the Adjacency Matrix, Adjacency List, and Edge List of this **specific** graph.

[Line limit **9**] Ans (you can answer horizontally instead of vertically, follow the template in ans_blank.txt):

MULTILINE
ANSWER

b) What is the **best** way to store a star graph of **V** ($1 \leq \mathbf{V} \leq 200\,000$) vertices (and obviously **V**-1 edges) if you are told that: the **star graph** (a "tree" with one internal vertex and **V**-1 leaves) has no edge weight but all vertices have weights (i.e., **vertex-weighted** graph). Moreover, you need to answer weight of any vertex and answer the existence of edge (u, v) both in O(1). An example of a vertex-weighted star graph with one internal vertex 0 and weight 3 (denoted as 0/3) is shown below, but you will need to answer for general vertex-weighted star graph.



[Line limit **5**] Ans:

SHORT
MULTILINE
ANSWER

Question 6 **[15 marks]**

There is an array of **N** (1 ≤ N ≤ 100 000) **strings** (each string is at most **M** ≤ 20 lowercase alphabet characters).
However, it is also known that we only need to print out **K** (1 ≤ **K** ≤ min(100, **N**)) shortest strings inside the array.
If there are more than one string that is equally short, print the ones that are alphabetically lower.
Propose the fastest (and correct) algorithm (with any necessary data structure) (either in pseudocode/English or in full C++/no bonus marks for doing this other than to strongly signal your live coding skill to the lecturer) to sort the array as per this requirement and **analyze its time complexity**.

You will get 0, 3, 6, 10, or 15 marks if your answer is wrong (max 1 partial mark for effort), "correct but O(**N**^{2} * **M**)", "correct but O((**N** log **N**) * **M**)", "only fast enough if **N** strings lengths are uniform between [1..20]", "correct and better time complexity than the other options, even if all **N** strings have length **M** = 20 characters", respectively.

Example: **N** = 5, **M** ≤ 5, **K** = 3, {"Andy", "Ben", "Cathy", "Donald", "Bob"} →, print: "Ben", "Bob", and then "Andy" because "Ben" and "Bob" are both 3-characters strings (and "Ben" is alphabetically lower than "Bob"), and then we also print "Andy" as it is a 4-characters string. Since we already get **K** = 3 strings that we want, we stop here.

[Line limit **9**] Ans:

```
1
2
3
4
MULTILINE
ANSWER
7
8
9
```

Question 7 **[20 marks, 4+6+4+6 = 20]**

Steven is stuck in his room, which is curiously modelled as a **rectangular R** x **C grid** (1 ≤ **R**, **C** ≤ 1000).
The rows are numbered from 0 to **R**-1 and the columns are numbered from 0 to **C**-1.
His kids have been more playful than usual and have left LEGO bricks on some grid cell(s) of your room.
Steven wishes to get to the door while trying his best to avoid the pain of stepping on LEGO brick(s).
He is currently at cell (R1, C1) and he wish to get to the exit door, which is at cell (R2, C2).
He can only walk from current cell to its North/East/South/West adjacent cell.
(He cannot walk diagonally and cannot exit the grid until encountering (R2, C2)).
Every time Steven steps on a cell that contains **X** LEGO brick(s) (**inclusive of (R1, C1) and (R2, C2)**),
he will always step on all **X** brick(s) on that cell and endure **X**-unit(s) of pain ☹.

Your task is to compute **path of least pain** (stepping on a LEGO brick is painful) for the four different constraints below. For each scenario, **propose the fastest (and correct) algorithm (with any necessary data structure)** (either in pseudocode/English or in full C++/no bonus marks for doing this other than to strongly signal your live coding skill to the lecturer) to compute the path of least pain given the constraints and **analyze its time complexity**.

a) Steven's room has **no obstruction** (so, Steven can step on any of the **R** x **C** cells), every cell has **exactly 1 (ONE)** LEGO brick (so, stepping on any cell causes Steven 1-unit of pain) (4 marks)

[Line limit **3**] Ans:

```
SHORT
AT MOST THREE LINES
ANSWER
```

b) Steven's room **has obstructions** (some cells cannot be stepped on, e.g., there are very expensive LEGO car in that cell and Steven doesn't want to take the risk of stepping on it and will avoid those "obstruction cells"), yet every other non-obstructed cell has **exactly 1 (ONE) LEGO brick** (6 marks)

[Line limit **5**] Ans:

```
SHORT
AT MOST FIVE LINES
ANSWER
```

c) Same as b (**has obstructions**), but now every non-obstructed cell curiously has **either 1 (ONE) or 0 (ZERO)** LEGO brick(s) (4 marks)

[Line limit **7**] Ans:

```
SHORT
AT MOST SEVEN LINES
ANSWER
```

d) Same as b (**has obstructions**), but now every non-obstructed cell has **0 (ZERO), 1 (ONE), or at most 7 (SEVEN)** LEGO brick(s) (6 marks)

[Line limit **9**] Ans:

```
SHORT
AT MOST NINE LINES
ANSWER
```

- **End of Paper** -