Automatic Grading

All questions in this section are automatically graded. The order of questions and the options are randomized. For fill-in-the-blanks, please follow the format as precisely as possible. Once you move beyond this section, you cannot go back.

1. Check all statements that are **True** with respect to the Single-Source Shortest Paths (SSSP) problems discussed in the Lecture 7b+8a.

(5 marks)

- We can easily (in not more than 15 lines) implement the O((|V|+|E|) log |V|) Modified Dijkstra's algorithm in Python because we can use Python's standard library: heapq in a Lazy manner.

  The original Dijkstra's cannot implement decreaseKey() operation effectively, while the modified Dijkstra's solve this.
- We can solve the **unweighted** SSSP problem using O(|V|+|E|) BFS algorithm.
- The O((|V|+|E|) log |V|) Modified Dijkstra's algorithm can be trapped in an infinite loop if we run it on a graph with negative weight cycle reachable from the source **s**.
- We can solve the **unweighted** SSSP problem using O((|V|+|E|) log |V|) Modified Dijkstra's algorithm.

  Unweighted graph can be solved by Modified Dijkstra's, but BFS is more effective
- We can solve the **non-negative weighted** SSSP problem using O((|V|+|E|) log |V|) Modified Dijkstra's algorithm.

All options are correct

2. Check all statements that are **True** with respect to the Traveling-Salesman-Problem (TSP) discussed in the last Lecture 8b.

(5 marks)

- The algorithm that tries all permutations of **N** vertices runs in O(**N!**) (or O(**(N-1)!**) if the first vertex is fixed to vertex 0).
- We can solve TSP using the **Modified Dijkstra's** algorithm to find the shortest tour that visits each vertex of the graph once. False

  Modified Dijkstra's cannot solve TSP effectively, which can only find the single-source shortest path, TSP requires a minimum loop include all nodes
- For small test cases, e.g., **N** <= 10, we can use Python's `permutations` function from `itertools` to help us try which permutation of **N** vertices yield the shortest cycle/tour.
- We can solve TSP using the **Breadth-First Search** algorithm to find the shortest tour that visits each vertex of the graph once. False

  Same problem as Dijkstra's, BFS can find shortest path between two nodes in a unweighted graph, but cannot find a minimum loop to travel all nodes

Only I and III are true

3. Check all statements that are **True** with respect to these Table ADT data structures discussed in Lecture 5a/5b.

(5 marks)

(Python custom-)implementation of Hash Table with Separate Chaining collision resolution technique is just a Python list of Python lists.

To avoid ambiguity, we are referring to HashTableDemo.py shown in class.

Hash Table is a list, and the element in Hash Table is also a list (often Doubly Linked List)

There are Table ADT a few additional operations that balanced BST implementation can do that a Hash Table implementation cannot do.

Balanced BST can find minimum/maximum element, check predecessor/successor

If we just need to support the 3 default Table ADT operations: Search(v), Insert(v), and Remove(v) efficiently (i.e., faster than O(**N**) per operation), we can *either* use a good Hash Table implementation (e.g., Separate Chaining) or a *balanced* Binary Search Tree implementation (e.g., self-balancing BST like AVL Tree - an extension of the standard BST that we learned in class).

Balanced BST take all 3 default operations as O(logN), Hash Table as O(1)

(Python custom-)implementation of Binary Search Tree is already good enough to be immediately used as a data structure to support Table ADT operations.

To avoid ambiguity, we are referring to BSTDemo.py shown in class (which is not yet an AVL Tree).

For normal BST, in some boundary cases it will like Linked List, operations will finished in O(N). But for most cases, BST is good enough.

All correct, option IV may vary depends on cases

4. Fill in the blanks

(5 marks)

You are given the following Python code. Notice the position of the comments.

```
N = 5000                # line 1
ans = 0                 # line 2
for i in range(N//2):   # line 3
    j = 1               # line 4
    while j < N:        # line 5
        j *= 3          # line 6
    for k in range(N): # line 7
        ans += 5        # line 8
print(ans)              # line 9
```

For all the boxes below, choose one of the following standard time complexities: **1**, log **N**, **N**, **N** log **N**, **N**^2, **N**^2 log **N**, or **N**^3. Do not write "O(" and ")" anymore as O(your-actual-answer) anymore that Big-O notation is already written. Remember that the grading by machine is **very strict**, so do not lose marks because of formatting issues.

The time complexity of **# line 1 to 2** (just before entering the outer loop) is O(__1__).

The time complexity of **# line 4 to 6** (the first part of the inner loop) is O(__2__).

The time complexity of **# line 7 to 8** (the second part of the inner loop) is O(__3__).

The overall time complexity of **# line 1 to 9 (i.e., the whole program)** is O(__4__).

The final value of ans when it is printed at line 9 is __5__.

Enter the correct answer below.

1			Character Limit: 3
2			Character Limit: 5
3			Character Limit: 5
4			Character Limit: 5
5		Please enter a number for this text box.	Character Limit: 9

5. Check all statements that are **True** with respect to the Graph Traversal algorithms: Depth-First Search (DFS) and Breadth-First Search (BFS) discussed in Lecture 6b/7a.

(5 marks)

- We can modify either DFS or BFS algorithm in order to find **ALL** topological orderings of a Directed Acyclic Graph (DAG) G = (V, E) in O(|V|+|E|) time.
- We run DFS from a source vertex **s** on a same graph G = (V, E) and it does *not* visit a vertex **x**. Therefore, if we run BFS from the same source vertex **s** on the same graph G = (V, E), it will also *not* visit vertex **x**.
- There is at least one graph traversal application that BFS can do but DFS cannot and vice versa.
- Both DFS and BFS **run in O(|V|+|E|)** when started from the same source vertex s on the same graph G = (V, E) which has been stored in an Adjacency List.
- Both DFS and BFS visit the same **set** of vertices when started from the same source vertex s on the same graph G = (V, E).
- Both DFS and BFS visit the same **sequence** of vertices when started from the same source vertex s on the same graph G = (V, E).

6. Check all statements that are **True** with respect to the Graph Data Structures discussed in Lecture 6a.

(5 marks)

- The best graph data structure to be used for **ALL** graph problems is Adjacency List.
- Adjacency List data structure uses O(|V|+|E|) space.

If the graph to be stored is a **dense** (to be precise, **near complete**) weighted graph, then using either Adjacency Matrix or Adjacency List will not be too much different, i.e., both will use O(V^2) space.

Converting a graph that is currently stored in an Adjacency Matrix into an Adjacency List takes O(|V|+|E|) time.

Adjacency Matrix data structure uses O(|V|+|E|) space.

7. Match the algorithm names on the left with its tightest worst-case time complexity. Assume that all graphs are simple graphs (no self-loop and no multi-edges).

(5 marks)

Drag the options on the left and drop them into the options on the right.

Depth-First Search on a graph G = (V, E)	O(log V)
	O(1)
Modified Dijkstra's on a non-negative weighted graph G = (V, E)	O((V+E) log V)
	O(V+E)
Breadth-First Search on a Tree T with V vertices	O(E log V)
	O(V)
Sort E weighted edges of an Edge List based on non-decreasing weights using Merge Sort	O(V log V)
	O(V^2)
Converting an Adjacency Matrix of a graph G = (V, E) into an Edge List	O(V^3)

Reset

8. Fill in the blanks

(5 marks)

Steven knows that most IT5003 students (final paper on 4 Dec 2021) will have attempted (parts of) CS2040C final paper on 27 Nov 2021, thus probably have seen the questions about median of 3 sorted arrays (called lists in Python), each array has the same length **N**.

You are given the following Python code that is supposed to solve that question, but is it the 'fastest way'? Let's find out:

```
>>> from statistics import median
>>> A, B, C = [1, 2, 3], [1, 5, 9], [2, 7, 8] # just an example
>>> median(A) # should be 2
2
>>> median(B) # should be 5
5
>>> median(C) # should be 7
7
>>> A+B+C # should be unsorted version of the 3 combined lists # question A
[1, 2, 3, 1, 5, 9, 2, 7, 8]
>>> median(A+B+C) # question B
3
>>> sorted(A+B+C) # to convince that 3 is the median of the 3 combined sorted lists
[1, 1, 2, 2, 3, 5, 7, 8, 9]
```

Now, the time complexity of the line `A+B+C` in terms of **N** as highlighted in `# question A` is O(__1__). Do not write O() anymore. Remember that `len(A) = len(B) = len(C) =` **N** for this problem.

Next, the time complexity of the line `median(A+B+C)` in terms of **N** as highlighted in `# question B` is O(__2__). Do not write O() anymore.

Note that for question B, we need to know what is inside the black-box of function `median`, so you are given the following information:

```
# Reference, Steven has unpacked the latest statistics.py of Python 3.10 from
# https://github.com/python/cpython/blob/3.10/Lib/statistics.py
# copied verbatim below:

# FIXME: investigate ways to calculate medians without sorting? Quickselect?
def median(data):
    """Return the median (middle value) of numeric data.
    When the number of data points is odd, return the middle data point.
    When the number of data points is even, the median is interpolated by
    taking the average of the two middle values:
    >>> median([1, 3, 5])
    3
    >>> median([1, 3, 5, 7])
    4.0
    """
    data = sorted(data)
    n = len(data)
    if n == 0:
        raise StatisticsError("no median for empty data")
    if n % 2 == 1:
        return data[n // 2]
    else:
        i = n // 2
        return (data[i - 1] + data[i]) / 2
```

Enter the correct answer below.

1 [ ] Character Limit: 7

2 [ ] Character Limit: 7

9. Check all statements that are **True** with respect to the various ADT (List/Stack/Queue/Deque/Priority Queue) topics discussed in Lecture 3a/3b/4a/4b.

(5 marks)

- We can implement Priority Queue ADT using Python heapq `H` efficiently (all operations in O(log **N**)) like this:

```
from heapq import heappush, heappop
def enqueue(v): heappush(H, v)
def peek(): return H[0]
def dequeue(): heappop(H)
```

- We can implement PriorityQueue ADT using Python list `L` efficiently (both `enqueue(v)` and `dequeue()` operations are in O(log **N**)) like the one shown in BinaryHeapDemo.py in class.

- We can implement Queue ADT using Python list `L` efficiently (all operations in O(**1**)) like this:

```
def enqueue(v): L.append(v)
def peek(): return L[0]
def dequeue(): L.pop(0)
```

- We can implement Stack ADT using Python list `L` efficiently (all operations in O(**1**)) like this:

```
def push(v): L.insert(0, v)
def peek(): return L[0]
def pop(): L.pop(0)
```

- We can implement Queue ADT using Python deque `D` efficiently (all operations in O(**1**)) like this:

```
from collections import deque
def enqueue(v): D.append(v)
def peek(): return D[0]
def dequeue(): D.popleft()
```

10. Check all statements that are **True** with respect to the Sorting topics discussed in Lecture 2a/2b.

(5 marks)

The underlying implementation of `L.sort()` to sort a Python list `L` is probably Timsort (a variant of Merge Sort).
We can sort a Python list `L` by calling `L.sort()` or `L = sorted(L)`
There is no difference between sorting a Python list `L` using either `L.sort()` or `sorted(L)`, i.e., list `L` will be sorted afterwards using either way.
The time complexity of sorting a Python list `L` containing **N** integers using either `L.sort()` or `L = sorted(L)` is O(**N** log **N**).
`L.sort()` is a stable sorting algorithm.

IQ Tests

Both questions in this section are a bit challenging. Do not burn too much time. There are two other application sections after this.

11. Steven claims that there is a better way to store a **complete unweighted** (simple) graph $\mathbf{K_N}$ with **N** vertices and **N* (N-1)/2** edges than the 3 default graph data structures discussed in class, i.e., not using Adjacency Matrix, Adjacency List, nor Edge List.

If you concur, explain the better way and analyze its space complexity or other benefits!

If you disagree, explain why one of the 3 default data structures is still the most appropriate way to store this kind of graph!

(5 marks)

Enter your answer here

Character Word Limit: 1000

12. Steven claims that there is a better way to sort (in ascending order) a nearly sorted list **L** with **N** non-negative and distinct integers than just using O(**N** log **N**) Python `L.sort()` or `L = sorted(L)` that has been asked in the earlier section (that you can now cannot go back to).

PS: Here, nearly sorted is formally defined as follows: Each of the integer is at most **K** indices away from its target sorted position and 0 <= **K** < min(**N**, 7). For example, see list **L** = [3, 1, 2, 4, 5] with **N** = 5 and **K** = 2. Notice that integer 3 (at index 0) is just 2 indices away from its sorted location (index 2). Similarly, notice that integer 1 and 2 are both just 1 index away from their sorted locations.

If you concur, explain the better way and analyze its time complexity!

If you disagree, explain why just sorting the whole **N** integers in O(**N** log **N**) using `L.sort()` or `L = sorted(L)` is already the best possible way!

(10 marks)

Enter your answer here

Character Limit: 2000

Application 1 (Second Last Section) - Queueing at Cashier Lines

A supermarket has **N** Cashiers. For each cashier, we know the number of people currently queuing to be served by that cashier. These people form a line (a queue), waiting to be served by that cashier.

Now, **M** customers will arrive in the next 1 minute (one by one, and during this next 1 specific minute, none of the cashier will finish processing the head of his/her line - so the line length will only increase). Each of these **M** customers (that come one after another in the next 1 minute) uses the same *greedy strategy*: queue at the back of the current shortest line (the line with the fewest number of people) and if ties, choose any such lines (it doesn't matter for this problem).

Your job is to determine the current number of people in the chosen cashier line every time each of the **M** customers join that cashier line.

The input contains two integers **N** and **M** in the first line (**N** and **M** as explained above), followed by **N** integers in the second line, denoting the number of people in each Cashier line initially. For example:

```
3 4
3 2 4
```

Means that there are **N** = 3 Cashiers (numbered Cashier 0, 1, and 2) and currently they are about to serve [3, 2, 4] customers, respectively.

We can visualize the initial state of the 3 Cashier lines as follows ('X's are customers already in each Cashier line):

```
Cashier Line = 0 | 1 | 2
-------------------------
               X | X | X
               X | X | X
               X |   | X
                 |   | X
```

Then, **M** = 4 customers come in the next 1 minute (one after another), so the first new customer (i.e., 'A') will greedily select Cashier 1 (**size 2 - we print 2**, now Cashier 1 size is 3), the Cashier lines become:

```
Cashier Line = 0 | 1 | 2
------------------------
               X | X | X
               X | X | X
               X | A | X
                 |   | X
```

Next, the second new customer (i.e., 'B') will then greedily select either Cashier 0 or 1 (both **size 3 - we print 3**, suppose he/she goes to Cashier 0, now Cashier 0 size is 4), the Cashier lines become:

```
Cashier Line = 0 | 1 | 2
------------------------
               X | X | X
               X | X | X
               X | A | X
               B |   | X
```

Note that even if 'B' goes to Cashier 1, we still print 3 at this stage, so there is no ambiguity.

Next, the third customer (i.e., 'C') will then greedily select the other line that the second customer doesn't select earlier (in this case, Cashier 1, of **size 3 - we print 3**, now Cashier 1 size become 4 too), the Cashier lines become:

```
Cashier Line = 0 | 1 | 2
------------------------
               X | X | X
               X | X | X
               X | A | X
               B | C | X
```

Finally, the last customer (i.e., 'D') will select any of the Cashier line (as all have **size 4 - we print 4**). Therefore, we output the following **M** = 4 lines:

```
2
3
3
4
```

13. Solve this problem in pseudo-code (you will need time to answer the longer last question after this). You can quote the name of some data structure(s) and/or algorithm(s) that we learned in class verbatim in your explanation (if nothing is modified). Analyze the worst-case time complexity of your proposed solution.

Note that you will only get full marks for this question if your proposed solution is correct and runs in O(**M** log **N**) --- so that **M** and **N** can be up to 50,000 (lots of customers and Cashier lines); partial if your solution is correct but runs in O(**MN**) --- so that **M** and **N** can be up to just 1000, or just very low mercy marks if your solution is actually incorrect.

Model Answer: 36 words and 259 characters only.

(15 marks)

Enter your answer here

Character Word Limit: 1000

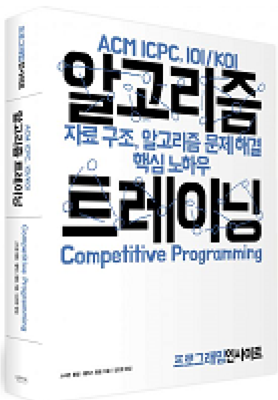
Application 2 (The Last Section) - Book Translation

Warning: This is the hardest question in this paper. If you don't spend enough time on this section, you may not be able to complete them on time.

=======

As most of you already know, Steven has written a book titled "Competitive Programming 4" (released on July 2020).

It has been translated into Korean (CP3) back in 2017.



In Spanish (CP4) back in early 2021 :).

And also in Bulgarian (CP3.18) recently (Nov 2021) :).

There are a few other book translation projects in the queue, e.g., Chinese, Portuguese, Japanese, and even Indonesian.

Although Steven can code in multiple *programming* languages (e.g., C++, **Python - this course**, Java, MATLAB, etc), he only speaks English and Indonesian (and too lazy to translate his own book to Indonesian).

So, Steven has enlisted (paid) translators to help translate between languages, but obviously they quote different costs. In some cases multiple translations might be needed. For example, if Steven can’t find a person who can translate his book from English to Swedish, but have one person who can translate from English to French and another from French to Swedish, then Steven can actually do the translation.

While minimizing the total cost of all these translations is important to you, the most important condition is to minimize each target language’s distance (in number of translation steps) from English, since this cuts down on the errors that typically crop up during any translation.

Fortunately, the Method to Solve (MtS) this problem is in Chapter 4 of Steven's book (this statement is True), so you should have no problem in solving this, right?

14. Input and potential (graph) data structure(s)

For this problem, the input starts with a line containing two integers **n** and **m** indicating the number of target languages and the number of translators that Steven has contacted (1 ≤ **n** ≤ 10,000, 1 ≤ **m** ≤ 50,000).

The second line will contain **n** short strings specifying the **n** target languages. Steven wants his book to be translated to *ALL* these **n** target languages.

After this line are **m** lines of the form **l1 l2 c** where **l1** and **l2** are two different languages (two short strings) and **c** (0 <= **c** <= 10,000) (let the unit be 1K SGD) specifying the cost to translate the book between these two languages (in either direction). The languages **l1** and **l2** are always either "`English`" (that has 7 characters) or one of the target (real-life) languages (all short language names will be not more than 20 characters), and any pair of languages will appear at most once in the input. The initial book is always written in "`English`".

A Sample Input looks like this:

```
4 6
Chinese French Portuguese Swedish
English Chinese 1
English French 1
English Portuguese 5
Chinese Portuguese 1
Portuguese Swedish 5
French Swedish 1
```

Write a simple Python code to handle the Input format as shown above and also show how you are going to store this graph information in a (graph) data structure(s). If you recall, the AM/AL/EL graph data structures that we learned in class only deal with vertices labeled from [0..|V|-1].

(7 marks)

Enter your answer here

Character Word Limit: 2000

15. Fill in the blanks

(3 marks)

Output and Sample Test Cases to Confirm Understanding

For each test case, your task is to compute **the minimum cost** to translate Steven's book (from "`English`") to **all** the **n** target languages, subject to the constraints described above, or print "`Impossible`" (case sensitive, notice capital "`I`") if it is not possible.

For this Sample Input 1 (shown earlier):

```
4 6
Chinese French Portuguese Swedish
English Chinese 1
English French 1
English Portuguese 5
Chinese Portuguese 1
Portuguese Swedish 5
French Swedish 1
```

The expected Output is:

```
8
```

Because Steven can translate "`English`" to "`Chinese`" for 1K SGD, "`English`" to "`French`" also for 1K SGD, then translate the "`French`" translation one more time into "`Swedish`" for another 1K SGD. Finally, for "`English`" to "`Portuguese`" into "Swedish" for another 1K SGD. Finally, for "English" to "Portuguese",

Steven prefers to pay 5K SGD (for translation accuracy) instead of going via "English" -> "Chinese" -> "Portuguese" that "only" costs 2K SGD. Thus, in overall, the total of the translation costs is 1+1+1+5 = 8K SGD. Thus, the output is "8".

=======

For this Sample Input 2:

```
2 1
A B
English B 1
```

The expected Output is:

```
Impossible
```

Because there is no suitable direct (or indirect) translator(s) to help us translate Steven's book into language "A".

=======

Now the questions that will be automatically graded. For the following Test Case A, B, and C, what should be the output?

Test Case A

```
5 5
AAA BBB CCC DDD EEE
English AAA 1
BBB English 2
DDD English 3
English EEE 5
English CCC 4
```

The output of Test Case A should be ___1___.

=======

Test Case B

```
5 5
AAA BBB CCC DDD EEE
English AAA 1
BBB AAA 2
DDD CCC 3
DDD EEE 5
BBB English 4
```

The output of Test Case B should be ___2___.

=======

Test Case C

```
2 3
BBB AAA
English AAA 10
BBB AAA 5
English BBB 14
```

The output of Test Case C should be ___3___.

Enter the correct answer below.

1 [ ] Please enter a number for this text box.

2 [ ]

3 [ ] Please enter a number for this text box.

16. Now the second last part, solve this problem in pseudo-code. You can quote the name of some data structure(s) and/or algorithm(s) that we learned in class verbatim in your explanation (if nothing is modified). Analyze the worst case time complexity of your proposed solution.

(7 marks)

Enter your answer here

Character Word Limit: 1000

17. Show the Python code of your previous answer (combining Q14+Q16 answers in Python code). Note that this part will be graded as 0 (regardless of what you wrote) if your previous answer in Q16 is incorrect. Thus, please explain your algorithm in pseudo-code first (in Q16) and only if you are convinced that your algorithm is correct and fast, then use the remainder of this final assessment to code the solution in Python for the last few marks.

Model Answer: 106 words and 725 characters only.

(3 marks)

Enter your answer here

Character Limit: 5000