UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
ESC190 FINAL ASSESSMENT – Apr. 24, 8:00am – Apr. 25, 8:00am
Estimated Time Required: 2.5 - 4 hours

Examiners: Saima Ali and Christopher Lucasius

Instructions

This is an open-book take-home assessment. Each section has multiple questions to choose from. Only answer the number of questions specified in the instructions as indicated in the table below. If you answer more than the indicated number of questions, we will only grade you on the first question(s) that appear in your uploaded solution. In addition to your Crowdmark submission, upload your code solutions to Quercus to the "Final Assessment" assignment as a .zip file.

For all programming solutions, external libraries are prohibited unless otherwise indicated.

We will be monitoring the **engsci.esc190@utoronto.ca** e-mail account for questions with the subject heading: *[ESC190] Final Assessment Question* during the following times (EDT): 9am-11am / 1pm-3pm / 5pm-7pm / 9pm-11pm

Links

Submit your written work and code in **one** document per question on CrowdMark:
https://app.crowdmark.com/sign-in/utoronto

Submit **only** your code (**one file per question**) on Quercus:
https://q.utoronto.ca/courses/129944/assignments/332839

Grade Breakdown

Section	# of Questions	Marks per Question	Total
1	2	5	10
2	2	5	10
3	2	5	10
4	2	10	20
5	1	20	20
6	1	30	30
Bonus 1	1	6	6
Bonus 2	1	4	4
			100

Permitted Aids

- C compiler (GCC/Clang), Valgrind, Python interpreter
    - You may SSH into the ECF machines or use the Virtual Box.
    - Versions: GCC 8.2.1/8.3.1, Clang 3.4.2/7.0.1, Valgrind 3.14.0, Python 3.6.8/3.6.9
- Linux terminal command `man`; e.g. `man fprintf`, `man malloc`
- Course materials (lecture slides/videos, notes, textbook, labs and solutions)

Unauthorized Aids

- Communication between peers
- Internet usage outside of the above permitted aids
- Online discussion platforms

You **must** sign the oath of academic integrity below, otherwise you will receive a "Did Not Write" mark on the final assessment.

Oath of Academic Integrity


Saima and Chris trust that you will not use any unauthorized aids for this final assessment. **You must sign this sheet to receive a grade in the final assessment.**

In submitting this assessment, I, _____, confirm that my conduct during this take-home exam adheres to the Code of Behaviour on Academic Matters. I confirm that I have not acted in such a way that would constitute cheating, misrepresentation, or unfairness, including but not limited to, using unauthorized aids and assistance, impersonating another person, and committing plagiarism. *I pledge upon my honour that I have not violated the Faculty of Applied Science & Engineering's Honour Code during this assessment.*

_____
Signature

_____
Date

Section #1 – Short Answer: Data Structures
Answer 2 of 3 questions.

(2 x 5 = 10 Marks)

Question #1.1

Alice is trying to pick a hash function for her hash table. She knows that all possible keys will be in the set K ⊆ ℤ (i.e. in the set of integers). She is deciding between:

$$h_1(x) = \text{floor}(\sqrt{|x|}) \bmod b$$
$$h_2(x) = |x| \bmod b$$

where b is the number of buckets, $x \in \text{K s.t.} -b \leq x \leq b$, the floor operator rounds down a number to the nearest integer, and mod is the modulo operator (% in Python/C). Assume you will not deal with duplicate keys (i.e. all keys are unique).

a) Indicate which hash function you would recommend to Alice and why.

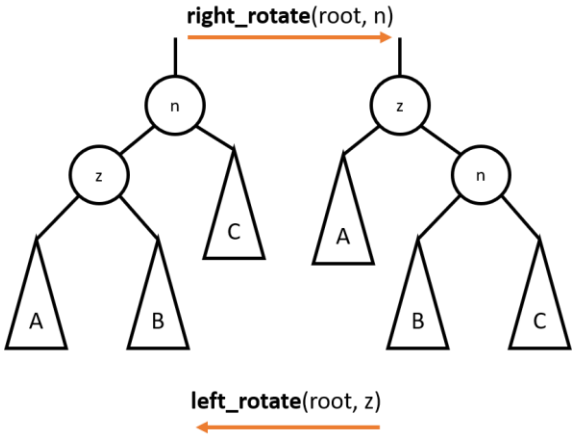
b) Indicate which of the following hash table modes you would recommend to Alice:

i. Closed addressing with chaining using linked lists
ii. Closed addressing with chaining using binary search trees
iii. Open addressing with linear probing
iv. Open addressing with quadratic probing

Justify your choice. Describe 2-3 relevant implementation considerations.

Question #1.2

Consider the two binary search trees (BSTs) below.



Suppose the balance factors of nodes `n` and `z` are $b_{n_{right}}$ and $b_{z_{right}}$, respectively, in the right BST and $b_{n_{left}}$ and $b_{z_{left}}$, respectively, in the left BST. Derive closed-form expressions for $b_{n_{left}}$ and $b_{z_{left}}$ in terms of $b_{n_{right}}$ and $b_{z_{right}}$ after a left rotation is applied to the right BST.

Question #1.3

We are looking to model the social network of the Ontario population (approximately 14 million people[1]). Each person has between 3-10 close friends and 1-8 immediate family members. An immediate family member is defined as a spouse, sibling, parent, or child.

a) Circle or indicate the type(s) of graph you will be working with for this problem:
   i. Directed / Undirected
   ii. Cyclic / Acyclic
   iii. Connected / Disconnected
   Define what the vertices and edges of your graph represents.

b) To implement this graph, indicate whether you would use an adjacency matrix or adjacency list and why. Choose an appropriate parameter and define the space complexity of your chosen implementation in tight-bound asymptotic notation.

c) Describe an algorithm that identifies whether two people are connected via a path with a depth $\leq \delta$, with $\Theta(\delta)$ worst-case space complexity. E.g. if Pichu is friends with **only** Ash, and Ash is friends with **only** Pichu and Brock, Pichu→Ash→Brock is a path of depth 2 connecting Pichu and Brock.

[1]https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=1710000901

Section #2 – Short Answer: Algorithms
Answer 2 of 4 questions.

(2 x 5 = 10 Marks)

Question #2.1

a) Jim, a tech-savvy librarian, was assigned to deal with a massive electronic record that he needs to sort in **non-decreasing** order by the end of the day. In a panic, he searches the Internet and copies over some code to run the standard implementation Quicksort on a **subset** of his records. The resulting records are sorted – but in **decreasing** order.

   i. What would you say to Jim about the practicality of copying Internet code blindly (i.e. "cargo-culting")? What would you suggest as an alternative?
   ii. Before Jim runs Quicksort on his **full set** of records, what are some considerations about the time efficiency of Quicksort he should be aware of?

b) Bob has crafted a novel comparison sorting algorithm and has called it bobsort.

   Bob claims: "Bobsort can sort a list of length n in worst-case in $\Theta(\log n)$ time."

   Using the following facts:
   i. Optimal comparison sorts run in worst-case $O(n \log n)$ time
   ii. Optimal comparison sorts run in worst-case $\Omega(n \log n)$ time
   iii. Bobsort is a comparison sort.
   iv. The tightest upper bound of the runtime of optimal comparison sorts is a tightest lower bound of the runtime of all comparison sorts, in the worst case.

   Construct a proof proving or disproving Bob's claim.

Question #2.2

Your calculator is broken, and you need to calculate the logarithm of a number x with a specific base. Implement an efficient algorithm in C that runs in $O(\log x)$ time to find the n^{th} base logarithm of a number $x \geq 1$ up to a certain tolerance ε. You may use **only** the following functions from math.h:

```
double pow(double base, double exp)
```
returns $base^{exp}$, i.e. $\Pi_{i=1}^{exp}\ base$

```
double fabs(double num)
```
returns $|num|$

```
double find_nth_log(double x, double n, double epsilon){
        /*
        x: number to take the logarithm of
        n: number specifying which logarithm base to apply
        epsilon: tolerance
        /*/
        // Implement below









}
```

Question #2.3

The following is pseudocode for a draft of the `is_max_heap` function:

```
function is_max_heap(vals):
    n ← length of vals
    i ← 0
    left_i = 2*i + 1
    right_i = 2*i + 2

    while i is not n:
        if vals[i] < vals[left_i]:
            return FALSE
        else if vals[i] < vals[right_i]:
            return FALSE
        endif
        i ← i+1
        left_i = 2*i + 1
        right_i = 2*i + 2

    endwhile
    return TRUE
end
```

If this code is implemented step by step in C, the code can be constructed to compile but a Valgrind error will occur.

a) Identify the step in the algorithm where the error would occur and what kind of Valgrind error will be present.

b) What error would this translate to in Python?

c) Based on your answers to (a) and (b), explain which programming language you would choose to implement this algorithm and why.

Question #2.4

The following pseudocode finds the m largest elements in a complete binary search tree with n nodes and stores them in `result`.

```
function m_largest(root, m):
    i ← 0
    result ← []
    limit_inorder(root)
end

function limit_inorder(node):
    if node and i < m:
        limit_inorder(node.right)
        if i < m:
            result.append(node.data)
        i ← i + 1
        limit_inorder(node.left)
end
```

a) How would you implement the variables `i` and `result` in C such that they can be used globally in all calls to `limit_inorder`?

b) Write down a recurrence relation representing the worst-case time complexity for `m_largest`.

c) Draw a recursion tree corresponding to the above recurrence relation and derive the worst-case time complexity for this algorithm in tight-bound notation.

Section #3 – Basic C
Answer 2 of 5 questions.

(2 x 5 = 10 Marks)

Question #3.1

`main.c`

```
#include <stdio.h>

int main(){
    char x = 1234567890000;
    printf("%d\n", x);
}
```

The provided code snippet, `main.c`, exists in your present working directory. The following two terminal commands are run:

```
gcc -o llama -Werror main.c
gcc -o aardvark main.c
```

a) What command would you run in the terminal to run the executable file compiled from `main.c`?

b) Explain the messages generated at compile time for the first command.

c) Based on your answer to (b), explain the difference between the first and the second command. Identify any changes you would make to `main.c` and why. Factually justify your answer.

Question #3.2

Examine the following code.

main.c

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    long long int i = 0;
    while (malloc(sizeof(int))){
        i++;
    }
    printf("%lli", i);
}
```

NOTE: This code may freeze your computer. It is not recommended that you run it with unsaved work open.

a) What Linux terminal command would you use to compile `main.c` to generate a target executable of `MALLOC_IS_FUN`?

b) Does this executable run Valgrind clean? Indicate how you know and explain why or why not referencing the contents of `main.c`.

c) Explain what the author of the code might have intended to do and why it accomplishes or fails to accomplish this functionality.

Question #3.3

You and your friend are working on a C coding project together (perhaps augmenting your ESC190 project for fun). Your friend uses Windows 10 and you use Linux CentOS. How will the use of different operating systems pose a challenge for your collaboration? Indicate an example of when this occurs for **each** of the following stages:

a) Writing the code.
b) Compiling the code.
c) Running the generated executable.

Question #3.4

You run the terminal command `ls` in your present working directory and see only the following files in your directory:

```
exam.h    exam.c    final.c   meme.c    uwu.c
```

a) An error message interrupts the process when running any one of the following commands:

```
gcc -o exam.o exam.c
gcc -o final.o final.c
gcc -o meme.o meme.c
gcc -o uwu.o uwu.c
```

Does `uwu.o` exist in the present working directory?

b) The following terminal commands are run without any errors:

```
gcc -o exam.o exam.c -c
gcc -o final.o final.c -c
gcc -o meme.o meme.c -c
gcc -o uwu.o uwu.c -c
```

Explain why the commands from part (a) generated errors, but the commands from part (b) did not.

c) Write a one-line compiler command to generate a target executable `FINAL_EXAM` using all the files ending in "`.c`". Indicate any restrictions or assumptions.

d) Assuming the commands from part (b) are run, indicate a different one-line compiler command from part (c) to generate an identical target executable `FINAL_EXAM` **without using** any "`.c`" files. Indicate any restrictions or assumptions.

Question #3.5

main.c

```
#include <stdio.h>

int main(){
    long long int x = 1234567890000;
    char y = (char) x;
    printf("%d\n", y);
}
```

Compile and run the above `main.c` file.

a) Indicate the compiler command you would use to compile this program. Do any of the following flags affect the compilation?

```
-Wall
-Werror
-pedantic
-Woverflow
```

b) Indicate any discrepancy between expected and actual behavior, and as a programmer, how to construct best practices to avoid this deviation.

c) What happens if we do **not** typecast `x` to a char? Are there any differences using the flags from part (a)? Why or why not?

Section #4 – Advanced C
Answer 2 of 3 questions.

(2 x 5 = 10 Marks)

Question #4.1

We would like to construct an L x M x N rectangular prism using a 3D array of integers in C. Note that L is the maximum value of the first dimension, M is the maximum value of the second dimension, and N is the maximum value of the third dimension. You must define these three values as macros.

In the following parts, you may **only** include `stdio.h`, and **you may not** use the address (&) operator.

a) Write a program that sets each element in a prism to 2^{i+j+k} where i is the index of its first dimension, j is the index of its second dimension, and k is the index of its third dimension.

b) Write a program that creates a second rectangular prism and copies all elements of the first prism into the second one. You can only use **one** for statement and no other loops within it.

c) Suppose we want to set all elements of the second prism at index a of its first dimension to 4^a. Write a program that does this using only **one** for statement and no other loops within it.

d) Suppose we want to set all elements of the second prism at indices b and c of its second and third dimensions, respectively, to 2^{bc}. Write a program that does this using only **one** for statement and no other loops within it.

Question #4.2

You are the proud CEO of AlgosGelatos, a chain of restaurants dedicated to supplying the best gelatos on your side of the globe. Its distribution system could use some automation, so you decide to write a C program to keep track of your stores and their gelatos. You may assume your initial chain of 100 stores sells 10 types of gelatos.

a) Write the definitions of three structures, `Gelato`, `AlgosStore`, and `AlgosGelatos`. They should contain the following information:

`Gelato`	`AlgosStore`	`AlgosGelatos`
• Name of flavour • Popularity number • Quantity	• Name of manager • Number of staff members • Array of gelatos	• Name of CEO • Array of AlgosStores

b) Suppose you are given a predefined `AlgosGelatos` variable `algo`. Write a function that iterates through each store and prints all its struct members. When printing the gelatos of each store, prepend a tab for readability.

c) Suppose your distributor is travelling with a fresh supply of gelatos, given in the form of an array of `Gelato`s (the `Quantity` member denotes how many of this type of gelato the distributor has in total). Write a function which takes in a pointer to an `AlgosGelatos` and an array of `Gelato`s and updates the `AlgosGelatos` structure such that the distributor gives an equal number of gelatos to each store. For each flavour, if there is a remainder of gelatos left, the distributor keeps them.

d) Suppose `AlgosGelatos` wants to expand its business by doubling its number of stores at any arbitrary point in time during the execution of the program. Modify your answer to part (a) to reflect this by changing only one member declaration.

Question #4.3

The year is 2038. That's a problem. Specifically, the Year 2038 problem. The Resistance is shaking because they've read:

> "The Year 2038 problem [...] relates to representing time in many digital systems as the **number of seconds passed** since 00:00:00 UTC on 1 January 1970 and storing it as a **signed 32-bit integer**. Such implementations cannot encode times after 03:14:07 UTC on 19 January 2038."[2]

Furthermore,

> "Programs that attempt to increment the time beyond this date will cause the value to be stored internally as a very large negative number, which these systems will interpret as having occurred at 20:45:52 on Friday, 13 December 1901 rather than 19 January 2038. This is caused by integer overflow, during which the counter runs out of usable digit bits and flips the sign bit instead."[2]

The time counter uses the most significant bit of the 32-bit integer as a "sign bit". For example:

$+10_{10}$

Value	**0**	0	0	…	0	0	1	0	1	0
Bit #	**31**	30	29	…	5	4	3	2	1	0

-10_{10}

Value	**1**	0	0	…	0	0	1	0	1	0
Bit #	**31**	30	29	…	5	4	3	2	1	0

a) Why is a **signed** data type used rather than an **unsigned** data type for this application?

b) The Resistance has acquired a computer which stores time using a 64-bit signed integer. Indicate when this computer will experience the "Year 2038" problem (hint: it will not be in year 2038). What time in the past will the computer be reset to? Be precise to the second – The Resistance is *counting* on you.

c) Write a C function which takes the input of the current system time in seconds after 00:00:00 UTC on 1 January 1970 as an n-bit signed integer, and checks if incrementing the time by 1 second will cause a "Year 2038" problem.

d) You may wish to use your work from part (c) here. Write a C function which takes as input:
   - Pointer to a dynamically allocated array of 8-bit chars (representing the current number of seconds elapsed since 00:00:00 UTC on 1 January 1970)
   - Current length of the array

   And as a result:
   - Modify the array by incrementing the value stored in the array by 1. Dynamically increase the size of the array by one element (i.e. 8 bits) if the increment would result in bit overflow error with the current array length.
   - Return the length of the modified array.

   E.g. Arrays of length 4 representing 32-bit signed integers:

$+10_{10}$

Value	**0**000 0000_2	0000 0000_2	0000 0000_2	0000 1010_2
Array index	3	2	1	0

-10_{10}

Value	**1**000 0000_2	0000 0000_2	0000 0000_2	0000 1010_2
Array index	3	2	1	0

[2] Quoted from https://en.wikipedia.org/wiki/Year_2038_problem

Section #5 – Long Answer
Answer 1 of 4 questions.

(1 x 20 = 20 Marks)

Question #5.1

A cipher is a tool to encrypt messages. The Caeser Cipher has a shift parameter, c, such that a character of the alphabet is mapped to the character c places down.

a) Fill out the table below for $c = 5$.

Letter	Encrypted ($c = 2$)	Encrypted ($c = 5$)
A	C	
B	D	
C	E	
. . .		
X	Z	
Y	A	
Z	B	
Message		
HELLO	JGNNQ	

b) Write a function in Python or C with the following specifications:
   i. Input parameters are `letter` and `c`.
   ii. Returns the encrypted value as per the Caeser Cipher

   Indicate any assumptions or conditions on the input values.

c) The v-length Vigenère Cipher applies a series of Caeser Ciphers of the form $\{c_1, c_2, c_3 \dots c_v\}$ to an n-length message where the ordered sequence $\{c_1 \dots c_v\}$ is called the cipher key. For example, if a 2-length Vigenère Cipher is applied with a key $\{4, 2\}$ to the message "HELLO":

Original Letter	H	E	L	L	O
Cipher Key	4	2	4	2	4
Encrypted Letter	L	G	P	N	S

Given an n-length encrypted message and its associated decrypted message, write a Python or C function which identifies and returns the v-length cipher key as per the Vigenère Cipher. Use your function from part (b).

For full marks, the cipher key you identify must be the smallest repeating block present in the identified key sequence ($v \leq n$). E.g.

Key sequence: $\{1, 2, 3, 1, 2, 3, 1, 2, 3, 1\}$
Cipher key: $\{1, 2, 3\}$

d) Consider the following scenario: you are given an encrypted message of length n. You are **not** given the decrypted message. However, you can verify whether a sequence of n characters is the decrypted message (i.e. TRUE or FALSE).

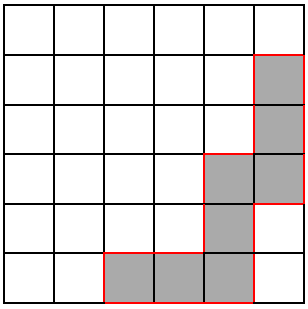
Suggest a step-by-step algorithm to decrypt the encrypted message given that it is encrypted with a Vigenère Cipher (with key length v) and derive an expression for the runtime complexity (in terms of n, v) of decrypting the encrypted message if:

i. You know the value of v
ii. You do not know the value of v

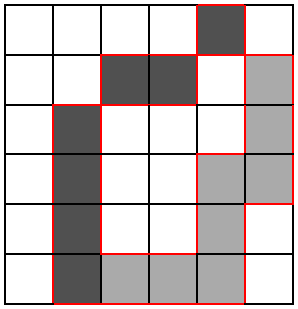
Question #5.2

Consider the following 2-dimensional arrays (left) and their resultant greyscale images (right).

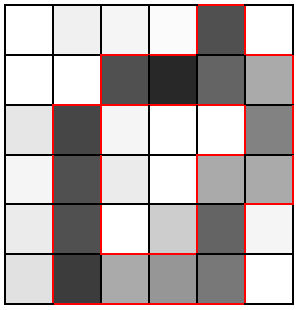
$$\begin{bmatrix} 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 170 \\ 255 & 255 & 255 & 255 & 255 & 170 \\ 255 & 255 & 255 & 255 & 170 & 170 \\ 255 & 255 & 255 & 255 & 170 & 255 \\ 255 & 255 & 170 & 170 & 170 & 255 \end{bmatrix}$$



$$\begin{bmatrix} 255 & 255 & 255 & 255 & 80 & 255 \\ 255 & 255 & 80 & 80 & 255 & 170 \\ 255 & 80 & 255 & 255 & 255 & 170 \\ 255 & 80 & 255 & 255 & 170 & 170 \\ 255 & 80 & 255 & 255 & 170 & 255 \\ 255 & 80 & 170 & 170 & 170 & 255 \end{bmatrix}$$



$$\begin{bmatrix} 255 & 240 & 245 & 251 & 80 & 255 \\ 255 & 255 & 80 & 40 & 100 & 170 \\ 230 & 70 & 245 & 255 & 255 & 130 \\ 245 & 80 & 235 & 255 & 170 & 170 \\ 235 & 80 & 225 & 205 & 100 & 245 \\ 225 & 60 & 170 & 150 & 120 & 255 \end{bmatrix}$$



a) Construct a mathematical definition for an edge which is outlined in red on each figure. Indicate any limitations of your definition.
b) Create a platform-independent algorithm (i.e. step by step instruction or pseudocode) to detect edges.
c) Implement your algorithm from (b) in Python or C and remark on its performance on each of the given example images.
d) Based on (b), explain how you might extend this to detect enclosed regions.
e) Based on (a) and (c), explain how you might address the limitation you identified.

Question #5.3

You are given the following MakeFile where only the dependencies are shown for your convenience:

```
file10: file6 file9
file9: file1 file7 file8
file8: file6
file7: file3
file6: file4 file5 file7
file5: file2 file3
file4: file2 file5
file3: file1 file2
file2: file0
file1: file0
```

Let each `file#` represent a source C file. Dependencies of each file follow the same syntax as a regular MakeFile.

a) Draw a directed graph to represent the above dependencies, where each edge connects a given file to one of its dependencies.

b) Write a linked list implementation of your graph above, either in Python or C.

c) Write an algorithm that returns a valid ordering of the above files for the compiler to run. Clearly write a valid ordering at the end of your answer.

d) Suppose the MakeFile rules are modified such that exactly one of the dependencies of each file must be run (i.e. not all). Write an algorithm that returns the total number of possible valid sequences of files that can be run given a start file and an end file. For example, three valid orderings between `file0` and `file10` are given below.

   i. `file0, file1, file9, file10`
   ii. `file0, file2, file4, file6, file10`
   iii. `file0, file2, file4, file6, file8, file9, file10`

Question #5.4

You are a bus driver about to lead a group of n families throughout the Savannah. Your cohort just arrived, and they are ready to board your bus. Before you can allow them on, you need to enter their **last names and ages** into your system. The kids, being the fastest and most eager, tell you their information first, followed by the teenagers, followed by the parents, followed by the grandparents, followed by the great-grandparents, etc. In short, younger members of the family are before older ones.

It's a long drive back, so the bus has to be as organized as possible. You need to sort your group such that the **following constraints** are met:

- All families (people with the same last name) are together.
- Younger members of a given family are placed closer to the back of the bus than older ones.

Unfortunately, a wild stampede of elephants is rushing towards you, so you need to sort them quickly. You also have no Internet, so you cannot download any sorting libraries.

a) Write an algorithm in Python or C to organize your cohort by sorting a given array of family information **in-place**. It should run in $\Theta(n \log n)$ time. You may assume the maximum length of a last name is 10 characters.

   Sample Input: [(Ledbetter, 5), (Freeney, 6), (Lutz, 4), (Freeney, 34), (Ledbetter, 40), (Lutz, 31), (Ledbetter, 60)]

   Sample Output: [(Freeney, 6), (Freeney, 34), (Ledbetter, 5), (Ledbetter, 40), (Ledbetter, 60) (Lutz, 4), (Lutz, 31)]

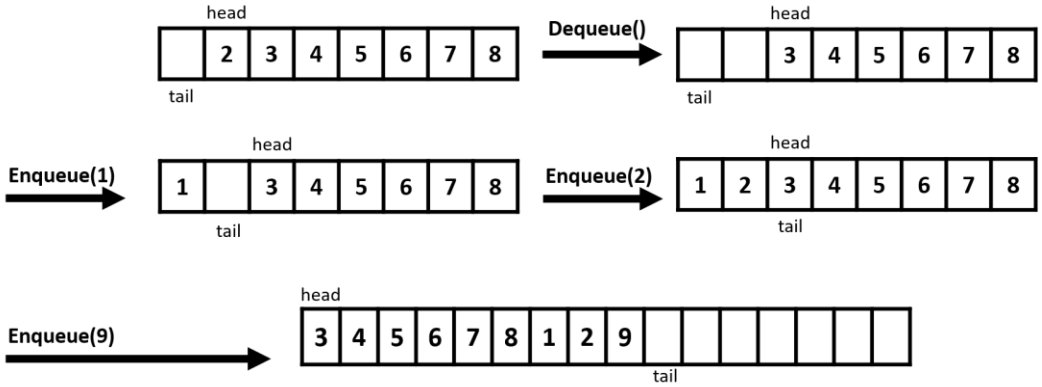
b) Write a derivation for the time and space complexity of your algorithm using trees and a recurrence relation.

Section #6 – Long Long Answer
Answer 1 of 4 questions.

(1 x 30 = 30 Marks)

Question #6.1

Suppose we want to implement a circular queue using an array that has an initial capacity (maximum number of elements) MAX. A circular queue is like a regular queue except that elements can be enqueued or dequeued by wrapping around it. **Assume we enqueue on the tail and dequeue from the head.** An example circular queue with sample operations is shown below:



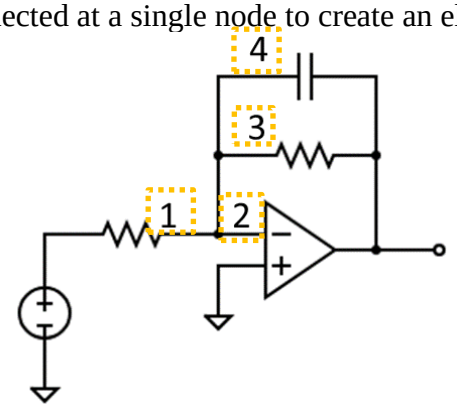
a) **Write a program in C that implements this circular queue while keeping track of its head and tail indices and number of elements in the queue.** If we enqueue while the queue is full, create a new queue double the size, insert all elements from the old queue into the new one, and enqueue the new element (see example above). If we dequeue while the queue is empty, print "Queue underflow". You may assume the following:

- The queue structure contains five members: array of nonzero integers, capacity, size, head, and tail
- An "empty" element is represented by a 0.
- Your program should contain functions to initialize a queue to MAX size, enqueue a nonzero integer, dequeue an integer from a queue and return it, and deallocate all memory taken up by the queue.
- You never need to shrink the capacity of the queue.

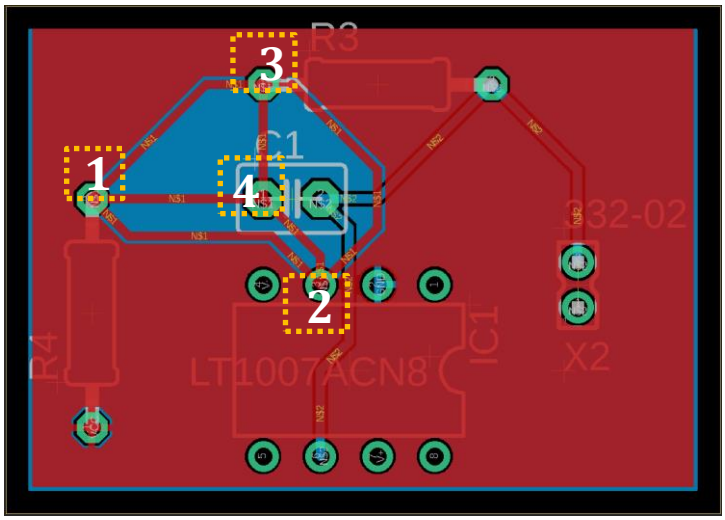
b) Suppose we implemented the circular queue using a linked list instead of an array. **How would this affect the time and space complexity of the enqueue and dequeue operations?**

Question #6.2

Consider the circuit schematic shown below. The terminals of the components labelled 1, 2, 3, and 4 must all be connected at a single node to create an electrical connection.



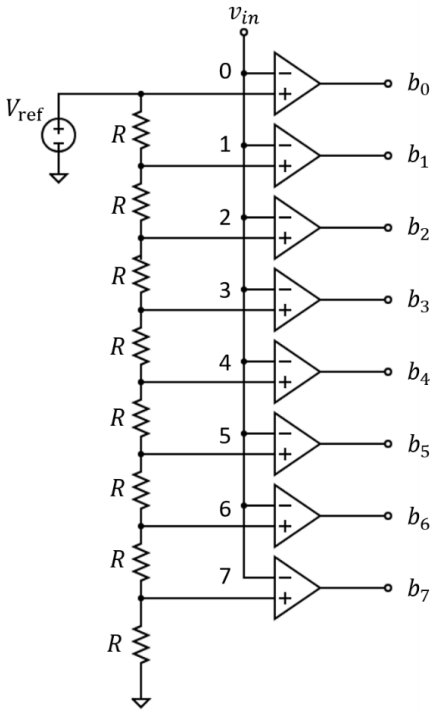
The design for a corresponding printed circuit board (PCB) for the schematic is show below, with labels 1, 2, 3, 4 corresponding to terminals of resistor R3, op-amp LT1007, resistor R4, and capacitor C1, respectively.



There are 6 possible connections, called **traces** (segmented lines in red), between the 4 highlighted component pins. The traces in the PCB have lengths in mil (thousandths of an inch), shown in the table below, e.g. the length of the trace connecting pins 1 and 3 is 383 mil. **You may assume the physical positions of the components are fixed**.

Pins	**1**	**2**	**3**	**4**
1	0	462	383	300
2		0	504	191
3			0	200
4				0

a) Draw a weighted graph representing the distances between each pair of pins.
b) Determine which pins should be connected (i.e. which of the 6 traces above should be manufactured) to each other to minimize the total trace length used in the PCB to connect them. Highlight the connections in the table above.
c) Write an algorithm in Python or C to determine the optimal connections in an arbitrary PCB that **minimizes** the **total trace length**.
d) Now consider the following schematic of an 8-bit analog to digital circuit below:



You just laid out all the above components onto a PCB. All components are connected except the 9 numbered pins (positive v_{in} pin and inverting input pins 0 - 7) shown in the schematic. You have recorded the following distances (in mm) between these pins in the table below (∞ means a trace cannot connect them). Determine which pairs of pins should be connected by highlighting the correct entries in the table below.

Pins	v_{in}	0	1	2	3	4	5	6	7
v_{in}	0	35	∞	∞	56	53	39	10	12
0		0	16	45	∞	43	44	55	18
1			0	40	31	78	∞	∞	8
2				0	∞	33	49	65	53
3					0	∞	34	52	62
4						0	20	∞	14
5							0	32	70
6								0	5
7									0

Question #6.3

Consider the matrices $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}$.

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{bmatrix}$$

The result of the matrix multiplication AB, given by Q, is:

$$Q = AB = \begin{bmatrix} \Sigma_{i=1}^{n} a_{1i} b_{i1} & \cdots & \Sigma_{i=1}^{n} a_{1i} b_{ip} \\ \vdots & \ddots & \vdots \\ \Sigma_{i=1}^{n} a_{mi} b_{i1} & \cdots & \Sigma_{i=1}^{n} a_{mi} b_{ip} \end{bmatrix}$$

a) Write the expression for an arbitrary entry of Q, q_{ij}, representing the entry in the i^{th} row and j^{th} column of Q. What is the dimension of Q?

b) Each matrix entry is of primitive C type ENTRY_TYPE. You store the matrices using **single-dimensional** dynamically allocated C arrays. Indicate how to index element q_{ij} for matrix Q. Suggest a suitable type for ENTRY_TYPE.

c) Construct an algorithm to perform matrix multiplication of 2 matrices using the format described in (b) uing the following tools:

- Loop structures (`for, while`)
- Conditional statements and expressions (`if, ==, <, >, &&, ||`)
- Bitwise operators (`&, |, ^, >>, <<`)
- Addition (+)
- Array indexing

You may **not** use:

- Multiplication (*) **outside of an array index operation**.
- Looping/recursion with addition to replicate multiplication.

d) Write a C function with the following input parameters:

- `ENTRY_TYPE *A`
- `ENTRY_TYPE *B`
- `char *size_A[2]`
- `char *size_B[2]`

which multiplies matrices A and B given the sizes in `size_A` and `size_B`. You may construct additional "helper" functions. Choose an appropriate return type.

e) Indicate the space and time complexity of your function from part (d). Indicate the variables that your analysis depends on and any assumptions that you make.

Question #6.4

One of Zeno's paradoxes involves the story of Achilles and a Tortoise who are racing along a one-dimensional racetrack. Achilles and the Tortoise are both moving in the positive direction. The Tortoise starts some finite positive distance ahead of Achilles. The Tortoise travels with a constant speed, and Achilles travels at a constant speed greater than that of the Tortoise.

Achilles paces himself using halfway points. Achilles sprints forward and manages to cover a distance $\frac{d}{2}$. He looks ahead and sprints the next $\frac{d}{4}$ ahead. The sequence of distances which he covers is $\frac{d}{2}, \frac{d}{4}, \frac{d}{8}, \frac{d}{16}$ ... where the i^{th} term $t_i = \frac{d}{2^i}$; $i \in [1, \dots, +\infty)$. Seemingly, Zeno's paradox suggests that Achilles can never overtake the Tortoise, because he can never overcome the distance d. However, we all know from MAT195 that the partial sum $S_n = \Sigma_i^n \left(\frac{1}{2^i}\right)$ converges to 1 as n tends to ∞.

a) Using the limit definition of a converging sum[3], i.e. $S_n = \Sigma_{i=1}^n t_i$ is said to converge if:

$$\exists s \in \mathbb{R} \text{ s.t. } |S_n - s| < \epsilon$$
$$\forall \epsilon > 0$$
$$\forall n \geq N, N \in \mathbb{Z} > 0$$

Create a step-by-step algorithm to determine if *any* series is convergent given the closed-form equation for the sequence terms t_i. When do you decide when it is appropriate to stop decrementing ϵ?

b) Implement your algorithm in Python or C. Ensure that you have a way to pass in a reference to the representation of the series terms (not hard coded in your main algorithm). You should **not** use `math.h`. Test your work on:

$$t_i = \frac{1}{2^i}\ (series\ converges)$$
$$t_i = \frac{1}{i}\ (series\ diverges)$$
$$t_i = \sin\left(\frac{i\pi}{4}\right)(series\ diverges)$$

c) What is the time complexity of your algorithm? How might you modify it such that:
   i. The time complexity is improved.
   ii. You identify the sum of the series, s, if the series converges.

[3] Referenced from https://en.wikipedia.org/wiki/Convergent_series

Bonus Section (10 Marks)

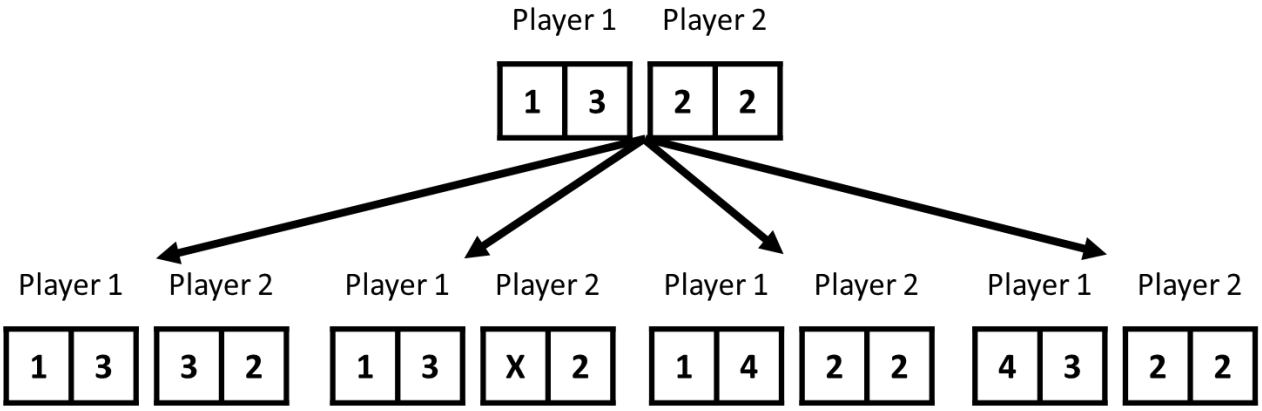
Answer any of the following questions.

Question #B1 (6 Marks)

Consider the following variant of the popular hand game of Chopsticks named Quick Sticks. The rules are as follows:

1. Each player starts the game with one finger on each hand.
2. On a player's turn, he/she can either use one hand to attack another player's hand or attack his/her other hand. The attacking hand adds its number of fingers to the attacked hand's number of fingers.
3. If any hand reaches greater than four fingers, that hand is out of the game.
4. The first player who loses both hands loses the game. The winner gains a score equal to the number of fingers left on his/her hand.

An example of four possible moves by Player 1 from an arbitrary state is shown below (each box represents a hand, each number represents the number of fingers on the hand, and X denotes a hand that is out of the game):



a) Draw the game tree for Quick Sticks from the initial state up to a depth of 2. You may represent each state by a group of four numbers (see above), each representing the number of fingers on the players' hands.

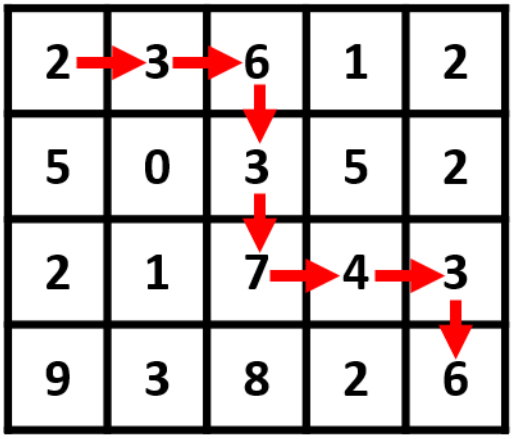
b) Design a program in Python or C to determine the optimal move (i.e. maximize final score) to make given an arbitrary state in Quick Sticks.

c) Determine the optimal move to make if you start the game.

Question #B2 (4 Marks)

You are searching for treasure in a two-dimensional map. Luckily, you have a radar that can map every location's treasure value (number of golden coins). You are located at the top left corner of the map, and you are able to move either right or down one unit at a time. Unfortunately, there is also a toll for travelling around the map, and you must pay 1 coin for every unit you move to the right and 2 coins for every unit you move down.

For example, one possible treasure value for the sample map shown below would be $2 + 3 - 1\ (\text{right}) + 6 - 1\ (\text{right}) + 3 - 2\ (\text{down}) + 7 - 2\ (\text{down}) + 4 - 1\ (\text{right}) + 3 - 1\ (\text{right}) + 6 - 2\ (\text{down}) = 24$.



a) Write an algorithm in Python or C that returns the maximum amount of treasure you can obtain after traversing an arbitrary map of treasure values. The input can be given in the form of a list of lists or array of arrays of treasure values. Determine the maximum amount of treasure which can be obtained from the map above.

b) Write another algorithm to determine which path you should take. This may be in the form of a list or pointer to int where each element is either 0 (go right one unit) or 1 (go down one unit). Determine an optimal path to take in the above map.

c) Derive an appropriate bound for the time and space complexity of your algorithm given an m x n map.