UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, APRIL 2024

DURATION: 2½ hours

ESC 190 H1S — COMPUTER ALGORITHMS AND DATA STRUCTURES

Calculator Type: None
Exam Type: B

Aids allowed: reference sheet distributed with the exam
Examiner(s): M. Guerzhoy

Student Number: |__|__|__|__|__|__|__|__|__|__|__|__|

UTORid: _____

UofT email: _____@mail.utoronto.ca

Family Name(s): _____

Given Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This final examination paper consists of 10 questions on 22 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy is complete, and fill in the identification section above.*

Answer each question directly on this paper, in the space provided. Use the pages at the end of the exam for extra space. If you use extra pages, indicate that you have done so in the space under the question.

Write up your solutions carefully! Comments and docstrings are *not* required to receive full marks, except where explicitly indicated otherwise. However, they may help us mark your answers, and part marks *might* be given for partial solutions with comments clearly indicating what the missing parts should accomplish.

When you are asked to write code, *no* error checking is required: you may assume that all user input and argument values are valid, except where explicitly indicated otherwise.

When writing in Python 3 programming language. You may not `import` any module except `math`, unless otherwise specified. When writing in C, You may `#include` any header file that is available with a standard installation of `gcc` or is described in the C99 standard, **unless otherwise specified**. We will accept code that is correct under the C99 standard as well as any code that will run correctly when compiled with `gcc`.

Write your student number on every odd-numbered page you use. Failure to do so will result in a penalty of 5 points.

MARKING GUIDE

# 1:	_____/ 10
# 2:	_____/ 15
# 3:	_____/ 15
# 4:	_____/ 10
# 5:	_____/ 15
# 6:	_____/ 10
# 7:	_____/ 5
# 8:	_____/ 10
# 9:	_____/ 5
# 10:	_____/ 5
TOTAL:	_____/100

Good Luck!

OVER...

Question 1. [10 MARKS]

Part (a) [8 MARKS]

Write a C function that computes the median of an odd number of integers. A median of an array of size n is a number m such that at least $(n - 1/2)$ elements of the array are smaller or equal to m and at least $(n - 1/2)$ elements of the array are larger or equal to m.

```
int my_median(int *arr, int sz)
{
```

Part (b) [2 MARKS]

What is a tight upper-bound on the asymptotic time complexity of the code you wrote? Simplify as much as possible. Briefly justify.

Question 2. [15 MARKS]

Write a function, `count_letters`, that counts the number of occurrences of each alphabetical letter found in a string. The function has two parameters: a string (i.e., `char *`) and an array of integers. The string should not be modified and can be of any length. You may assume that the string is null-terminated and all letters are lower-case. The string may contain characters that are not part of the alphabet (e.g., `0, 1, !, &,` etc). The integer array has a size of 26, one for each letter in the alphabet. The first index corresponds to the letter a, the second index to the letter b, and so on.

```
void count_letters(char *s, int counts[])
{
```

Question 3. [15 MARKS]

In C, write a function that takes a string as input and reverses the order of words in that string. A word is defined as a continuous sequence of non-space characters, and words are separated by one or more spaces. The function should return a new string, leaving the input string unchanged.

For example, given the string `"Hello EngSci Hi"`, your function should return `"Hi EngSci Hello"`.

You may assume that the input only contains letters and spaces.

```
char* reverse_words(char *str)
{
```


CONT'D...

Question 4. [10 MARKS]

In C, write a function that takes in an array of integers and its size, and returns 1 if the array is strictly increasing (e.g., `{1, 5, 10, 12}`) and 0 otherwise (e.g., `{1, 2, 2, 3}` or `{5, 4}`). You **must** use recursion. You **must not** use loops, global variables, or any helper functions.

```
int is_increasing(int *arr, int sz)
{
```

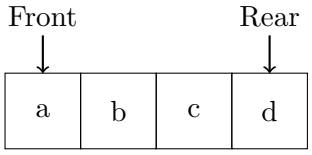
Question 5. [15 MARKS]

In C, implement a **circular queue** of strings.

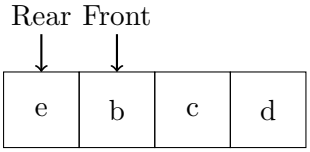
Your implementation must include ENQUEUE and DEQUEUE operations, which must work with enqueing and dequeing C strings. Your implementation should enable the use of enqueue an arbitrary number of elements.

A *circular queue*, is a data structure that uses a an array and two pointers to indicate the start position and the end position.

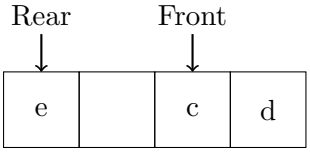
Consider a *circular queue* with a maximum size of 4, that holds elements 'a', 'b', 'c', and 'd'. The initial state of the queue might be:



After a dequeue operation, the front pointer moves to 'b'. After an enqueue operation of 'e', the rear pointer moves to 'e'. The queue now contains: 'b', 'c', 'd', and 'e'.



If another dequeue operation is performed, the front pointer moves to 'c'. The queue becomes: 'c', 'd', 'e', and the front position is now free. This illustrates the efficient use of space in a *circular queue* where elements can wrap around to the beginning of the array.



Question 6. [10 MARKS]

Part (a) [5 MARKS]

Consider a modified version of the game of Race to 21.

- Players take turns to count, starting from 1.
- Each player, on their turn, can count up 1, 2, or 3 numbers.
- The player who says "21" first is the winner.

  Here's an example of how this game could be played:

```
Start at: 0
Player 1: 1, 2
Player 2: 3, 4, 5
Player 1: 6, 7
Player 2: 8
Player 1: 9, 10, 11
Player 2: 12
Player 1: 13, 14
Player 2: 15, 16, 17
Player 1: 18, 19
Player 2: 20
Player 1: 21 (Player 1 wins)
```

Write a function that takes in the starting position (e.g., 0 as in the example above, or e.g. 12), and prints out the move from that position that will result in a guaranteed win (if it is available) which will take the smallest amount of moves even if an opponent tries their best to delay loss.

You must use depth-first search.

You may use C or Python.

CONT'D...

Part (b) [5 MARKS]

If you did not use recursion in the previous question, solve the same problem with recursion. If you did use recursion, solve the same problem without using recursion. You still must use depth-first search.

CONT'D...

Question 7. [5 MARKS]

Use A* to try improve the performance in the previous questions.

Question 8. [10 MARKS]

Given a non-empty string s and a dictionary `wordDict` containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

Write a function `canBeSegmented(s, wordDict)` that returns `true` if s can be segmented and `false` otherwise.

Example

1. If you are given:

   s = "applepenapple",
   `wordDict` = ["apple", "pen"],
   The output should be `true` because "applepenapple" can be segmented as "apple pen apple".

2. If you are given:

   s = "catsandog",
   `wordDict` = ["cats", "dog", "sand", "and", "cat"],
   The output should be `false` because "catsandog" cannot be segmented into a space-separated sequence of one or more dictionary words.

Notes

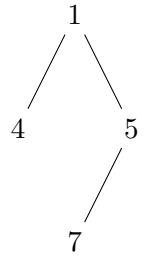
- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

You must use dynamic programming.

Question 9. [5 MARKS]

Part (a) [2 MARKS]

Construct a binary search tree as shown. You must use the code below. Store the root of the tree in the variable `root`.




```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

Part (b) [3 MARKS]

Write a function that takes in the root of a binary search tree and returns the sum of all the nodes in the tree.

Question 10. [5 MARKS]

Part (a) [2 MARKS]

In C, write an appropriate hash function for storing `double`s in a hash table.

```
unsigned int hash(double d)
{
```

Part (b) [1 MARK]

Briefly explain why you expect that the hash function you wrote would work in general.

Part (c) [2 MARKS]

Describe a dataset of `double`s that would cause the hash table using your hash function to perform poorly. Your description should include pseudocode of how to generate such a dataset.

Extra space for solutions

CONT'D...

Extra space for solutions

Extra space for solutions

CONT'D...

Extra space for solutions

Extra space for solutions