UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, APRIL 2016

CSC 190 H1S — COMPUTER ALGORITHMS AND DATA STRUCTURES

Exam Type: A — NO calculator allowed
Examiner(s): Pirathayini Srikantha

Student Number: |_|_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

Lecture Section: ☐ LEC 01 (Mon 11 am - 12 pm) ☐ LEC 02 (Mon 5 pm - 6 pm)

---

*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 5 questions on 21 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the final examination is complete and fill in the identification section above.*

Answer each question directly on the examination paper, in the space provided. If you need more space for one of your solutions, use one of the extra "blank" pages at the end of the examination and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! In particular, use notation and terminology correctly and explain what you are trying to do—part marks *will* be given for showing that you know the general structure of an answer, even if your solution is incomplete.

When writing code, comments are not required, although these may help us mark your answers.

MARKING GUIDE

# 1:	_____/ 35
# 2:	_____/ 15
# 3:	_____/ 15
# 4:	_____/ 15
# 5:	_____/ 20
TOTAL:	_____/100

Good Luck!

OVER...

Question 1. [35 MARKS]

This section consists of 20 short answer questions. Please fill in your answers in the space provided under each question. You can list your answers concisely in point-form.

Part (a) [1.5 MARK] List one advantage and one disadvantage of defining a variable as a specific type (e.g. `unsigned short int`, `long int`, etc. instead of regular `int`).

Part (b) [1.5 MARK] Is there a potential problem with the following code snippet? Why or why not?

```
int func(float a)
{
   float b=a/6;
   return b;
}
```

Part (c) [1.5 MARK] What is the general idea behind the *divide and conquer* problem solving technique?

Part (d) [1.5 MARK] What are the three issues in the following code snippet?

```
int * func( float a)
{
   int * b = (int *)malloc(sizeof(int));
   int c;
   *b=a+2;
   c=*b;
   return &c;
}
```

Part (e) [1.5 MARK] Declare a function pointer variable named `fPtr` that points to an already defined function handle named `sampleFunc` that takes in two input arguments which are both integers and returns a float value.

Part (f) [1.5 MARK] What does the following code snippet achieve?

```
int func(int a, int n)
{
   return (a & (1<<n));
}
```

Part (g) [1.5 MARK] When a structure variable is dynamically allocated, is it necessary to also dynamically allocate the corresponding members?

Part (h) [1.5 MARK] Suppose that an algorithm has a cost $f(n) \in O(g(n))$. State the formal definition of the O-notation in terms of $f(n)$ and $g(n)$ and include a graph to illustrate this definition.

Part (i) [1.5 MARK] What is the insertion/removal policy for queues? What is the insertion/removal policy for stacks? Which one of these linear data structures will be better suited for processing postponed obligations?

Part (j) [1.5 MARK] What is a common structural attribute between heaps and AVL trees?

Refer to Fig. 1 for the next 4 sub-questions. The root of each tree is the top-most node.

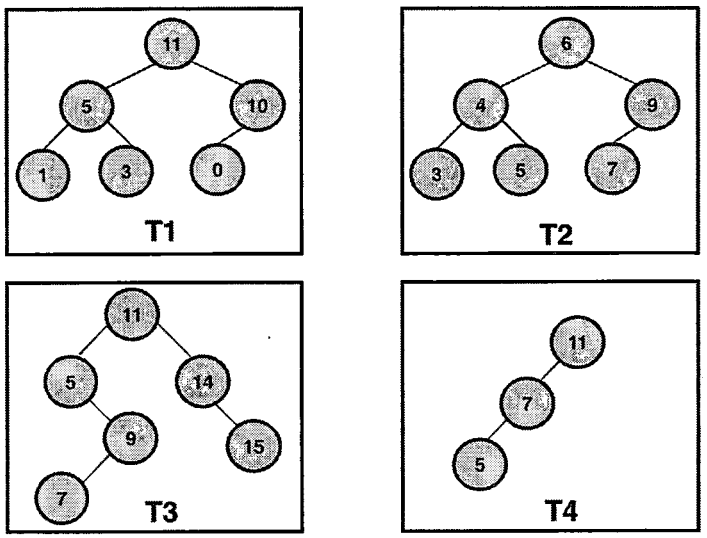


Figure 1: Trees

Part (k) [1.5 MARK] Provide an answer for each of the following based on T1:

- Is T1 a heap, BST and/or an AVL tree?
- What is the height of the tree?
- What is the result of an in-order traversal of the tree?

Part (l) [1.5 MARK] Provide an answer for each of the following based on T2:

- Is T2 a heap, BST and/or an AVL tree?
- Is this a complete and/full tree?
- What is the result of a pre-order traversal of the tree?

Part (m) [1.5 MARK] Provide an answer for each of the following based on T3:

- Is T3 a heap, BST and/or an AVL tree?
- What is the depth of the node containing the value 9?
- What is the result of a post-order traversal of the tree?

Part (n) [1.5 MARK] Provide an answer for each of the following based on T4:

- Is T4 a heap, BST and/or an AVL tree?
- Which nodes have the same degree?
- What is the result of a level-order traversal of the tree?

Refer to Fig. 2 for the next 2 sub-questions. The root of each tree is the top-most node.

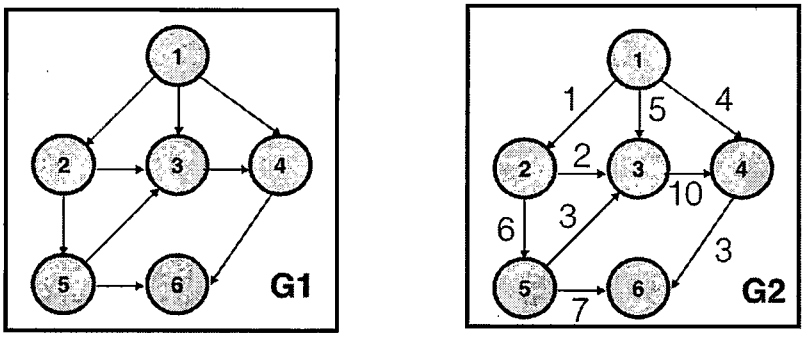


Figure 2: Trees

Part (o) [1.5 MARK] What is the topological ordering of nodes in G1?

Part (p) [1.5 MARK] What edges compose the edges of a minimal spanning tree in graph G2?

Part (q) [5 MARK] What is the algorithmic complexity of the following code snippet in big-oh notation? You may assume that $n = 2^k$ where k is a positive integer. A formal proof is not necessary but can help with part marks if your answer is incorrect.

```
void f (int n){
   int j = 1;
   while (j <= n){
         n=n/2;
   }
}
```

Part (r) [1.5 MARK] In a hash table, what degree of occupancy will guarantee constant time search and insertion into the hash table?

Part (s) [3 MARK] Draw the step by step break-down and combination of the following array when mergesort is applied to sort elements in the array. A=[9,8,13,2,3,10,7]

Part (t) [1.5 MARK] What is a disadvantage of using quicksort over mergesort?

Question 2. [15 MARKS]

Implement the function `insertBST` that inserts a new node into a binary search tree (BST) which contains an integer data `value`. Ensure that the insertion preserves the BST ordering. Following is the definition of the node structure used in the BST:

```
struct Node {
    int data;
    struct Node * lChild;
    struct Node * rChild;
};
```

Each new node should be dynamically allocated. The function `insertBST` takes in two arguments. The first argument is the pointer to the `root` (this can be `NULL` when the tree is empty at the beginning prior to any insertion). The second argument is the `value` to be stored in the `data` member of the new node to be inserted into the BST tree. This function returns the pointer to the root of the tree after insertion (this is necessary especially when the tree is initially empty). *Hint: Recursive implementation is straightforward.*

```
struct Node * insertBST(struct Node * root, int value)
{

}
```


CONT'D...

Question 3. [15 MARKS]

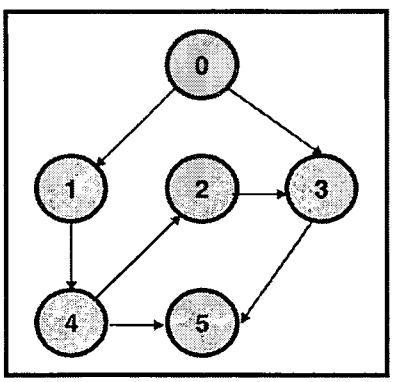
Implement the function `bfTraversal` which performs breadth-first traversal of a directed graph represented by `aM` starting from the node labelled `n` and prints out the labels of nodes as these are being traversed. Note that the provided underlying graph representation is an *adjacency matrix*. You may assume that the following definitions and declarations are provided to you:

```
#define NODES 6;
struct adjMat
{
   int mat[NODES][NODES];
   int visitedNodes[NODES];
};
struct Queue * initQueue();
void enqueue(struct Queue * q, int node);
int dequeue(struct Queue * q);
int isEmpty(struct Queue * q);
```

`NODES` represents the total number of vertices in the graph. Labels assigned to vertices in the graph take values in the set [0, NODES-1]. The structure `adjMat` provides the underlying graph representation via the adjacency matrix `mat`. The entry `mat[i][j]` is set to 1 if there is an edge in the graph from the vertex labelled `i` to the vertex labelled `j` and 0 otherwise. `visitedNodes` member in the `adjMat` structure is an array that keeps track of the nodes that have already been visited. You can assume that the `adjMat` structure is populated with appropriate values representing the graph before being passed as an argument to the function `bfTraversal`.

The queue interface functions can be used as helper functions for the breadth-first traversal implementation. You may assume that these interface definitions are available (i.e. you do not have to implement these). `initQueue` dynamically allocates a queue structure. `enqueue` inserts a new node into the queue containing information about the label of a vertex in the graph. `dequeue` removes a node from the queue and returns the vertex label information stored in the removed node. `isEmpty` returns 1 if there are no nodes in the queue and 0 otherwise.

As an example, consider the graph illustrated in the following figure. If the initial starting node `n` is 0, then the function `bfTraversal` should traverse the graph starting from node labelled 0 in breadth-first order and print out the nodes as these are being traversed as follows: `0 1 3 4 5 2`.



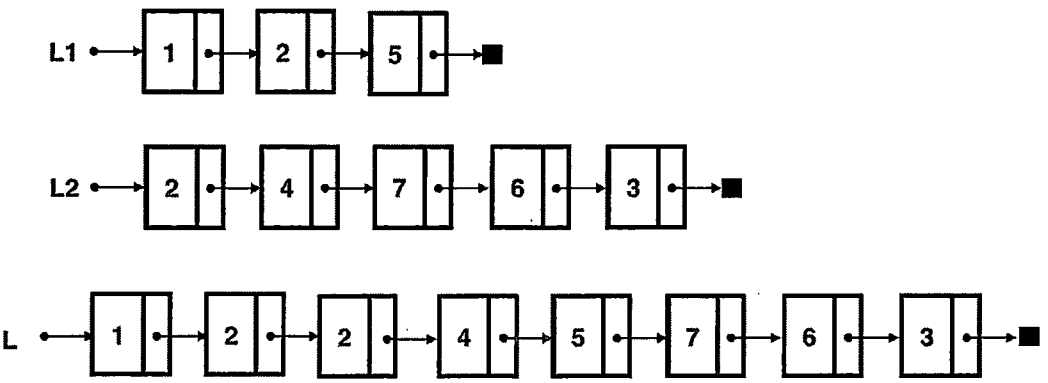
```
void bfTraversal(struct adjMat * aM, int n)
{

}
```

CONT'D...

Question 4. [15 MARKS]

Given pointers `L1` and `L2` to two linked lists with nodes that are dynamically allocated, implement the function `mergeLinkedLists` which combines both linked lists **in-place** (i.e. do not create new nodes) so that nodes in `L1` and `L2` alternate with one another (starting with the first node in `L1`, then the first node in `L2`, and so on). The pointer to the resulting merged linked list is returned by this function. In your implementation, ensure that there are no memory leaks and appropriate checks for NULL values are included. When one list is longer than the other, append all remaining nodes in the longer list to the final list when alternation is no longer possible. Following is an example of two linked lists `L1` and `L2` and the merged linked list `L` returned by the function.



Following is the definition of the node structure used in the linked lists:

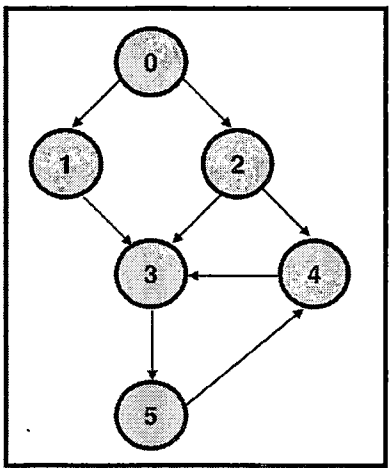
```
struct Node{
   int data;
   struct Node * next;
};
```

```
struct Node * mergeLinkedLists(struct Node * L1, struct Node * L2)
{

}
```

Question 5. [20 MARKS]

Consider a directed graph represented by an *adjacency list* `aL`. Each node in the provided graph is assigned a label from the set `[0,NODES-1]` where `NODES` is the total number of vertices in the graph. In this question, you will implement a function that computes the total number of vertices in the longest non-cyclic path starting from node `n`. Consider the following example. One of the longest non-cyclic path starting from node 0 is 0-1-3-5-4 and there are 5 vertices in this path. 5 is the value that should be returned by the function `longestPathLength` when `n=0`. *Hint: This is similar to computing the height of a tree. Additional measures must be taken to account for cycles introduced in graphs and the possibility of more than two descendants per vertex in the graph.*



You may assume that the following definitions are provided. You may define additional helper functions if needed.

```
#define NODES 6
struct Node{
    int data;
    struct Node * link;
};
struct adjList{
    struct Node * ptrArray[NODES];
    int visitedNodes[NODES];
};
```

The structure `adjList` provides the underlying graph representation where the member `ptrArray` contains the adjacency set of every node in the graph (i.e. `ptrArray[i]` points to the adjacency set of the vertex labelled `i`). The adjacency set of each node is represented by a linked list which consists of nodes each defined by the structure `Node`. The `data` member stores the vertex label information and `link` points to the next node in the linked list. The `visitedNodes` member of the structure `adjList` keeps track of vertices in the graph that have been visited.

```
int longestPathLength(struct adjList * aL, int n)
{

}
```

CONT'D...

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

Aid Sheet

- **Arithmetic sequence:**

$$S = \sum_{i=0}^{n-1}(a + id) = \frac{n}{2}(2a + (n-1)d)$$

- **Geometric sequence:**

$$S = \sum_{i=0}^{n-1} ar^i = \frac{a(1-r^n)}{1-r}$$

- **Heaviside cover-up** rules for partial fraction expansion:

$$\frac{ax^2+bx+c}{(x-d)(x-e)^2} = \frac{A}{x-d} + \frac{B}{x-e} + \frac{C}{(x-e)^2}$$

$$A = \frac{ad^2+bd+c}{(d-e)^2} \quad C = \frac{ae^2+be+c}{(e-d)}$$

B : **solve for B** $\quad \frac{ac^2+bc+c}{(c-d)(c-e)^2} = \frac{A}{c-d} + \frac{B}{c-e} + \frac{C}{(c-e)^2}$

where c is a constant