Instructions

This is a proctored, timed final exam covering content you have studied in Modules 1-8 (Focus on modules 1-6).

This exam contains:

15 multiple choice questions (2 points each)

2 fill in the blanks questions (10 points each)

4 coding questions (10 or 15 points each)

- Make sure you log in to our Zoom classroom for proctoring.
- Your exam will be invalid (0 points) if you do not log in!
- You will have 180 minutes (3 hours) to complete the exam from the time you first access the exam.
- Once you've opened the exam you must complete it within that time.
- You will not be able to log out and continue at another time.
- Also, make sure you do not quit the window containing the exam.

Allowed:

- Python, C/C++, Java compilers/interpreters
- IDEs such as Visual Studio Code, Pycharm, IntelliJ, Eclipse, etc..
- Textbooks
- Notes
- Source code
- Not allowed:
- Communicating with someone else (via Internet or otherwise)
- Using internet to search for answers (such as slashdot)
- ChatGPT
- If you are having trouble posting code and retaining the format, click on the Paragraph menu in the HTML editor and select Preformatted.

OR

Include your file as an attachment instead. Select Insert->Document->Upload Document.

Question 1

Which of the following has a big O(n^2)?
- 2 log n
- n + 4n^2 (ans)
- n^2 + n^4
- n^3 + n

Question 2

You have two data structures: one is a singly linked list with a tail reference, and the other is a doubly linked list WITHOUT a tail reference. For which operation would the singly linked list be more efficient?
- Search for item
- Delete last item
- Prepend item
- Append item (ans)

Question 3

Which sorting algorithm is the slowest if an array is sorted in reverse order?
- Quicksort
- Bubble Sort
- Merge Sort
- Selection Sort

Question 4

What is the time complexity of the following recursive Fibonacci function?

```
def fib(n):
  if n < 2:
    return n
  return fib(n - 1) + fib(n - 2)
```

==> ans: O(2^n)

Question 5

Which factor affects the performance of using quicksort on an array the most?
- The order of the elements in the array
- Choosing the pivot value (ans)
- The number of duplicate elements in the array
- Which factor affects the performance of using quicksrt on an array the most?

Question 6

In what situation will a Hash Table lookup get close to a time complexity of O(n)?
- When the hash table has a large number of buckets
- Too few collisions
- Too many collisions (ans)
- Too much memory allocated

Question 7

For which operation is binary search trees more efficient than sorted, ordered arrays?
- Searching
- Updating
- Insertion(ans)
- Indexing

Question 8

When performing Depth First Traversal and Breadth First Traversal, which data structures are usually used respectively?
- Stacks and queues (ans)
- Stacks and arrays
- Arrays and stacks
- Queues and stacks

Question 9

What is TRUE about adjacency lists and adjacency matrices?
- Edge lookup is O(1)
- Both are space efficient for dense graphs
- Weight lookup is O(1)
- Efficient when adding a new edge

Question 10

Which statement below about heaps is FALSE?
- Weakly ordered
- A node can have an empty left child but have a right child (ans)
- Finding the last node must be efficient
- Must be a complete tree

Question 11

In Dijkstra's algorithm, when does the algorithm terminate?
- When the priority queue is empty (ans)
- After a fixed number of iterations
- When all vertices have been visited
- When it reaches the destination vertex

Question 12

When deleting a word in a Trie, what kind of traversal should you use?
- Preorder
- Any order will do
- Postorder (ans)
- Inorder

Question 13

How does Kahn's Algorithm (Topological Sort) work?
- Builds a path based on the in-degree count of each vertex
- Keeps track of vertices in a stack (ans)
- Counts the outgoing edges of each vertex

- Uses a variation of Dijkstra's Algorithm to find a path

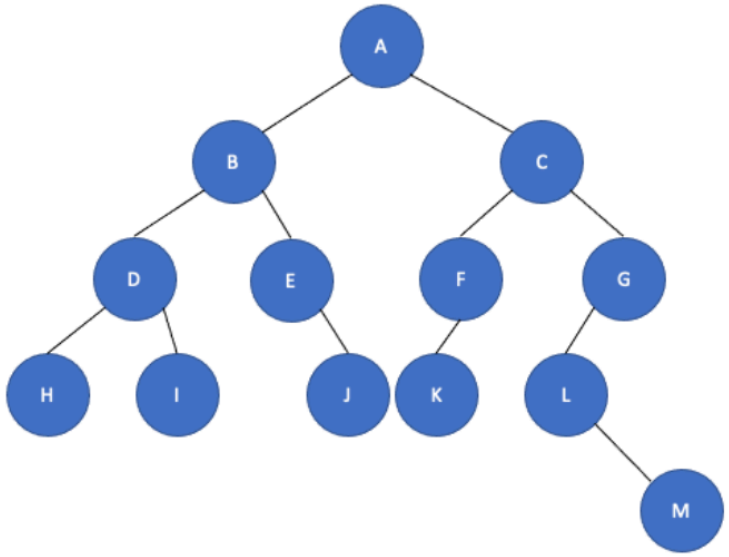
Question 14

Which technique generally uses the most space?
- Tail recursion
- Top-down recursion (ans)
- Iteration
- Greedy algorithm

Question 15

Which concept does not take advantage of bitwise storage and/or bitwise operators?
- Unicode
- Knapsack Problem Solution (ans)
- Huffman Coding
- Floating Point Representation

Question 16

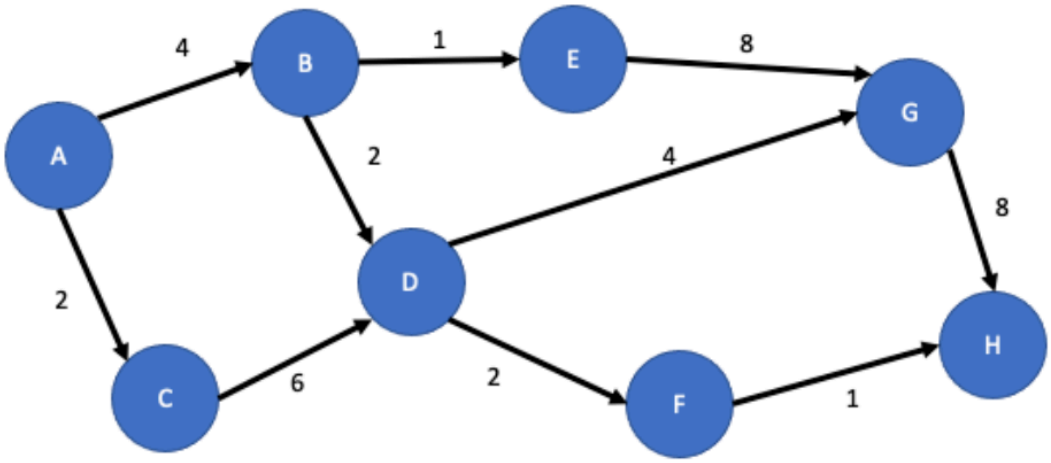


Given the binary tree above, list the order of node values for:
1. Preorder traversal
2. Inorder traversal
3. Postorder traversal
4. Breadth first traversal

Preorder should be: A B D H I E J C F K G L M
Inorder should be: H D I B E J A K F C L M G
Postorder should be: H I D J E B K F M L G C A
Breadth first traversal: A BC DEFG HI JK LM

Question 17

Use Dijkstra's Algorithm to determine from vertex A to vertex H:
1. The weight table (Use **insert->table** from the menu)
2. The shortest path



The weight table

	A	B	C	D	E	F	G	H
A	0	4	2	6	5	8	10	11
B	0	0	0	2	1	4	6	9
C	0	0	0	6	0	8	10	18
D	0	0	0	0	0	2	4	3
E	0	0	0	0	0	0	8	16
F	0	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0	8
H	0	0	0	0	0	0	0	0

the shortest path: A->B->D->F->H (weight: 9)

Question 18

Write your code in C/C++, Java or Python.
Write a function to check if linked list has a cycle.

My answer: use two pointers, and the second pointer runs twice faster as the first pointer. When the two pointers meet, that means the linked list has a cycle. If the two pointers didn't meet, means there has no cycle in this linked list.

Python code:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def hasCycle(head):
    if not head or not head.next:
        return False

    slow = head
    fast = head.next

    while fast and fast.next:
        if slow == fast:
            return True
        slow = slow.next
        fast = fast.next.next

    return False

# Creating a linked list with a cycle
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = head.next  # Creating a cycle

print(hasCycle(head))  # Output: True
```

Question 19

Write your code in C/C++, Java or Python.
Write a function that will accept a binary tree and return a node-by-node copy of it. The copy should have the same values and structure.

Python code:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def copyBinaryTree(root):
    if not root:
        return None
```

```
    new_root = TreeNode(root.val)
    new_root.left = copyBinaryTree(root.left)
    new_root.right = copyBinaryTree(root.right)

    return new_root

# Example usage:
# Constructing a sample binary tree
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)

# Creating a deep copy of the binary tree
copied_tree = copyBinaryTree(root)

# Testing if the copy is successful
print(copied_tree.val)  # Output: 1
print(copied_tree.left.val)  # Output: 2
print(copied_tree.right.val)  # Output: 3
print(copied_tree.left.left.val)  # Output: 4
print(copied_tree.left.right.val)  # Output: 5
```

Java Code:

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class DeepCopyBinaryTree {
    public TreeNode copyBinaryTree(TreeNode root) {
        if (root == null) {
            return null;
        }

        TreeNode newRoot = new TreeNode(root.val);
        newRoot.left = copyBinaryTree(root.left);
        newRoot.right = copyBinaryTree(root.right);

        return newRoot;
    }

    // Helper method to print the binary tree (in-order traversal)
    public void printBinaryTree(TreeNode root) {
        if (root != null) {
```

```
            printBinaryTree(root.left);
            System.out.print(root.val + " ");
            printBinaryTree(root.right);
        }
    }

    public static void main(String[] args) {
        DeepCopyBinaryTree solution = new DeepCopyBinaryTree();

        // Constructing a sample binary tree
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);

        // Creating a deep copy of the binary tree
        TreeNode copiedTree = solution.copyBinaryTree(root);

        // Testing if the copy is successful
        solution.printBinaryTree(copiedTree);  // Output: 4 2 5 1 3
    }
}
```

Question 20

Write your code in C/C++, Java or Python.
Write a function that accepts a valid binary heap and verifies whether it is a max-heap.

Java

```
def isMaxHeap(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        left_child = 2 * i + 1
        right_child = 2 * i + 2
        if left_child < n and arr[left_child] > arr[i]:
            return False
        if right_child < n and arr[right_child] > arr[i]:
            return False
    return True

# Example usage:
# Define a binary heap as an array
heap = [9, 7, 6, 5, 4, 2, 1]

print(isMaxHeap(heap))  # Output: True
```

Python

```
public class MaxHeapVerifier {
    public static boolean isMaxHeap(int[] arr) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--) {
            int leftChild = 2 * i + 1;
```

```
            int rightChild = 2 * i + 2;
            if (leftChild < n && arr[leftChild] > arr[i]) {
                return false;
            }
            if (rightChild < n && arr[rightChild] > arr[i]) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        // Define a binary heap as an array
        int[] heap = {9, 7, 6, 5, 4, 2, 1};

        System.out.println(isMaxHeap(heap)); // Output: true
    }
}
```

Question 21

Write your code in C/C++, Java or Python.
Write a function that accepts a starting vertex in a graph and returns the count of all the other vertices it can reach. You may use either a adjacency list or adjacency matrix graph implementation.

Python:

```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        self.graph[u].append(v)

    def dfs_count_reachable_vertices(self, start):
        visited = set()

        def dfs(node):
            visited.add(node)
            for neighbor in self.graph[node]:
                if neighbor not in visited:
                    dfs(neighbor)

        dfs(start)
        return len(visited) - 1  # Excluding the start vertex itself

# Example usage:
graph = Graph()
graph.add_edge(0, 1)
graph.add_edge(0, 2)
graph.add_edge(1, 2)
```

```
graph.add_edge(2, 0)
graph.add_edge(2, 3)
graph.add_edge(3, 3)

start_vertex = 2
print(graph.dfs_count_reachable_vertices(start_vertex))  # Output: 3
```

Java:

```
import java.util.*;

class Graph {
    private Map<Integer, List<Integer>> adjList;

    public Graph() {
        this.adjList = new HashMap<>();
    }

    public void addEdge(int u, int v) {
        adjList.putIfAbsent(u, new ArrayList<>());
        adjList.get(u).add(v);
    }

    public int dfsCountReachableVertices(int start) {
        Set<Integer> visited = new HashSet<>();
        dfs(start, visited);
        return visited.size() - 1; // Excluding the start vertex itself
    }

    private void dfs(int node, Set<Integer> visited) {
        visited.add(node);
        List<Integer> neighbors = adjList.getOrDefault(node, new
ArrayList<>());
        for (int neighbor : neighbors) {
            if (!visited.contains(neighbor)) {
                dfs(neighbor, visited);
            }
        }
    }

    public static void main(String[] args) {
        Graph graph = new Graph();
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 2);
        graph.addEdge(2, 0);
        graph.addEdge(2, 3);
        graph.addEdge(3, 3);

        int startVertex = 2;

System.out.println(graph.dfsCountReachableVertices(startVertex)); //
Output: 3
    }
}
```