# CSC 202, Summer 2023, Final Exam

**Name: ______________________**

- Write the answers in the space provided in boxes for multiple choice questions.
- It’s a close book, close notes test.
- You can’t use computers, tablets or any kind of tech for this test.
- You may use all forms that you know from your homework.
- If you need a method and you don’t know whether it is provided, define it.
- If you need extra space, use extra space at the back side of paper.

Good luck!

## Q1. Multiple Choice Questions (15x1 points)

1. Which sorting algorithm works by repeatedly inserting an element into a sorted portion of the list?
   a) Selection Sort
   b) Insertion Sort
   c) Merge Sort
   d) Quick Sort

2. Which sorting algorithm is most efficient for sorting a large collection of elements?
   a) Selection Sort
   b) Insertion Sort
   c) Merge Sort
   d) Quick Sort

3. In hashing _________ converts a key into an array index.
   a) Hashing value
   b) Index value
   c) Mapping value
   d) Hash function

4. Which data structure is implemented by hashing in Python?
   a) Array
   b) Linked List
   c) Stack
   d) Dictionary

5. Which collision resolution technique in hashing uses linked lists to store collided elements?
   a) Linear Probing
   b) Quadratic Probing
   c) Chaining
   d) Rehashing

6. In insertion sort, what is the maximum number of comparisons required to sort a list of n elements in the worst case?
   a) n
   b) n^2
   c) n log n
   d) n(n-1)/2

7. In merge sort, what is the time complexity for merging two sorted arrays of sizes n and m?
   a) O(n)
   b) O(m)
   c) O(n + m)
   d) O(n * m)

8. Incomplete code snippet:

```
def func(hash_table, key, value):

        hash_value = hash(key)

        while hash_table[hash_value] is not None:
 # Incomplete code

         hash_table[hash_value] = value
```

```
hash_table = [None] * 10
func(hash_table, "apple", 3)
func(hash_table, "banana", 7)
func(hash_table, "cherry", 5)
print(hash_table)
```
What should be added at the "Incomplete code" line to handle collision in linear probing?
a) hash_value = (hash_value + 1) % len(hash_table)
b) hash_value = (hash_value + 1) // len(hash_table)
c) hash_value = (hash_value - 1) % len(hash_table)
d) hash_value = (hash_value - 1) // len(hash_table)

9. What is the missing code in the following snippet to complete the Quick Sort algorithm?
```
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i + 1

def quick_sort(arr, low, high):
    if low < high:
        # Missing code: _________
        quick_sort(arr, low, pi - 1)
       quick_sort(arr, pi + 1, high)
# Usage:
numbers = [5, 3, 1, 4, 2]
quick_sort(numbers, 0, len(numbers) - 1)
print(numbers)
```

a) pi = partition(arr, low, high)
b) quick_sort(arr, high, low)
c) quick_sort(arr, pi + 1, high)
d) quick_sort(arr, low, high)

10. Suppose you have the following list of numbers to sort: [11, 7, 12, 14, 19, 1, 6, 18, 8, 20] which list represents the partially sorted list after three complete passes of selection sort?
a) [7, 11, 12, 1, 6, 14, 8, 18, 19, 20]
b) [7, 11, 12, 14, 19, 1, 6, 18, 8, 20]
c) [11, 7, 12, 14, 1, 6, 8, 18, 19, 20]
d) [11, 7, 12, 14, 8, 1, 6, 18, 19, 20]

11. Suppose you are given the following set of keys to insert into a hash table that holds exactly 11 values: 113 , 117 , 97 , 100 , 114 , 108 , 116 , 105 , 99 Which of the following best demonstrates the contents of the hash table after all the keys have been inserted using linear probing?

a) 100, __, __, 113, 114, 105, 116, 117, 97, 108, 99
b) 99, 100, __, 113, 114, __, 116, 117, 105, 97, 108
c) 100, 113, 117, 97, 14, 108, 116, 105, 99, __, __
d) 117, 114, 108, 116, 105, 99, __, __, 97, 100, 113

12. For a successful search using open addressing with linear probing, the average number of comparisons is approximately.
    a) $\lambda$
    b) $\lambda/2$
    c) $\frac{1}{2}(1 + \frac{1}{1-\lambda})$
    d) $\frac{1}{2}(1 + (\frac{1}{1-\lambda})^2)$
13. For a unsuccessful search using open addressing with linear probing, the average number of comparisons is approximately.
    a) $\lambda$
    b) $\lambda/2$
    c) $\frac{1}{2}(1 + \frac{1}{1-\lambda})$
    d) $\frac{1}{2}(1 + (\frac{1}{1-\lambda})^2)$

14. Which collision resolution technique in hashing uses linked lists to store collided elements?
    a) Linear Probing
    b) Quadratic Probing
    c) Chaining
    d) Rehashing

15. Suppose you have the following list of numbers to sort:
    [15, 5, 4, 18, 12, 19, 14, 10, 8, 20] which list represents the partially sorted list after three complete passes of insertion sort?
    a) [4, 5, 12, 15, 14, 10, 8, 18, 19, 20]
    b) [15, 5, 4, 10, 12, 8, 14, 18, 19, 20]
    c) [4, 5, 15, 18, 12, 19, 14, 10, 8, 20]
    d) [15, 5, 4, 18, 12, 19, 14, 8, 10, 20]

Q2. A student has proposed the following method as a faster way of counting the number of elements in a doubly-linked list with a sentinel head node. (2 points)

```
def fast_count(self):
    n = 0
    h = self.head.next
    while h is not self.head:
        if h.next is not self.head:
            n, h = n+2, h.next.next
        else:
            n, h = n+1, h.next
    return n
```

What is the time complexity of fast_count when run on a list with N elements? Just write big O notation. No need of explanation.

Q3. Consider the following code for implementing minimum heap. Complete the following missing code. (10 points)

```
class MaxHeap:
  def __init__(self):
    self.heap = []
  def __str__(self):
    return ' '.join([str(i) for i in self.heap])
  def parent(self, i):
    return (i - 1) // 2
  def left_child(self, i):
    return 2 * i + 1
  def right_child(self, i):
    return 2 * i + 2
  def swap(self, i, j):
    self.heap[i], self.heap[j] = self.heap[j], self.heap[i]
  def insert(self, data):









  def extract_max(self):
    if not self.heap:
        raise IndexError("Heap is empty")
    max_element = self.heap[0]
    last_element = self.heap.pop()
    if self.heap:
        self.heap[0] = last_element
        self.heapify(0)
    return max_element

  def heapify(self, i):
```

```
# Example usage:
max_heap = MaxHeap()
max_heap.insert(12)
max_heap.insert(10)
max_heap.insert(15)
max_heap.insert(20)
print(max_heap)  # Output: 20 12 15 10
print(max_heap.extract_max())  # Output: 20
print(max_heap)  # Output: 15 12 10
```

```
Q4.        8(B)                                                                        (6 points)

         /          \

     4(B)           12(B)

      / \            /     \

 2(R)  6(R)     10(B)  14(R)

         \

        7(R)
```

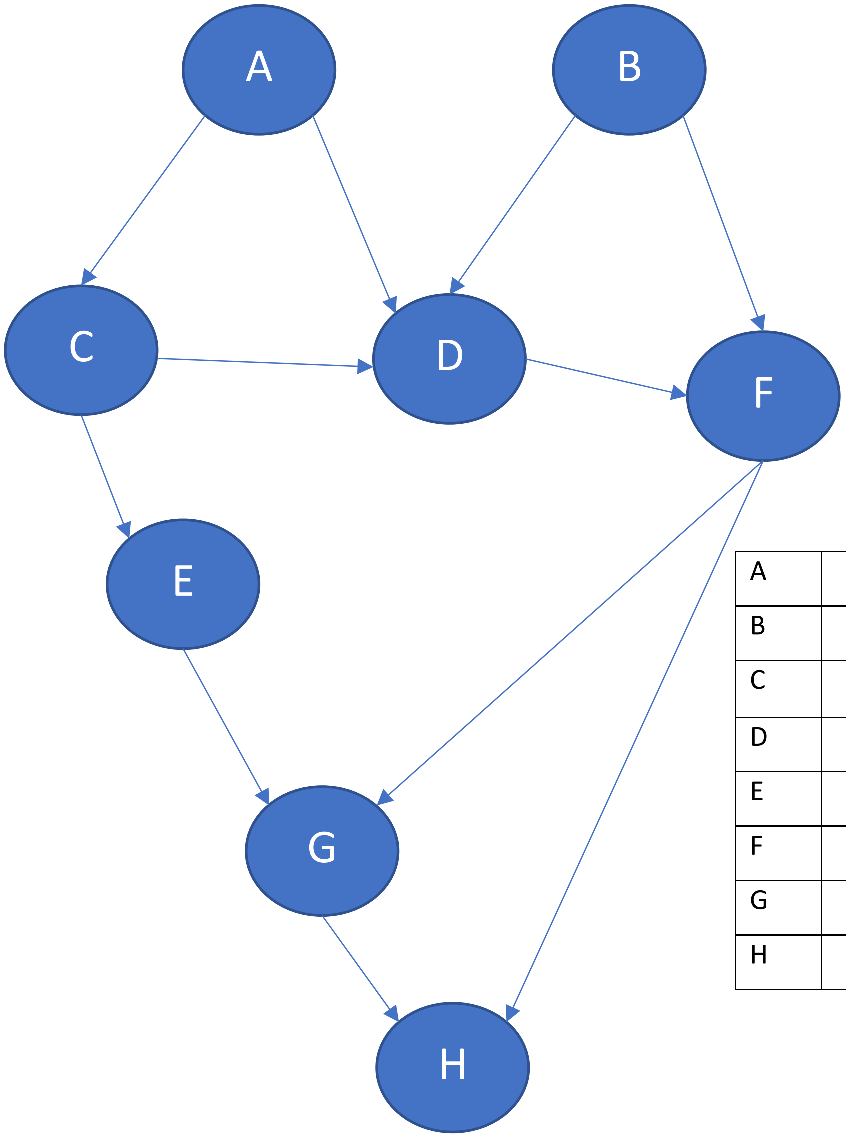
You are given this Red-Black Tree. Now, perform the following operations and provide the updated Red-Black Tree after each operation:

1. Insert the value 5 into the tree.
2. Delete the value 12 from the tree.
3. Insert the value 15 into the tree.

Please provide the Red-Black Tree structure after each operation and make sure it satisfies all Red-Black Tree properties.

Q5. Draw a picture of the Binary Search Tree after inserting the numbers 26, 16, 31, 32, 13, 18, 28, 29, 17 in the order given into the BST and then deleting the number 26. Assume the immediate successor will be placed into the location of the deleted node (4 points)

Q6. Give the adjacency list representation for the above graph. (8 points)



| | |
|---|---|
| A | |
| B | |
| C | |
| D | |
| E | |
| F | |
| G | |
| H | |

Q7. Problem: Managing Parkinson's Patient Data with AVL Tree (2 points)

Parkinson's disease is a neurodegenerative disorder that affects a person's motor functions. Patients with Parkinson's require regular monitoring and data management to track their symptoms and progress over time.

You are tasked with designing a data management system for Parkinson's patients using an AVL (Adelson-Velsky and Landis) tree. Each node in the AVL tree represents a patient record with the following attributes:

- Patient ID (a unique identifier)
- Patient Name
- Age
- Date of Diagnosis
- Severity Level (a measure of the disease's progression, where higher values indicate more severe symptoms)

Write the data definition for the above problem.

Q8. By using the problem described in Q7, write steps to implement the following operations: (2x5=10 points)

1. Insert a Patient Record: Add a new patient record to the AVL tree while maintaining the AVL property (balance factor of -1, 0, or 1 for each node).
2. Delete a Patient Record: Remove a patient's record from the AVL tree based on their ID.
3. Update Severity Level: Given a patient's ID, update their severity level in the AVL tree.
4. Search for Patients: Given a range of severity levels (e.g., mild, moderate, severe), search the AVL tree to find all patients who fall within that range.
5. Display Patients by Severity: Display all patients in the AVL tree in sorted order of severity levels.

Your AVL tree should automatically re-balance itself after each insertion or deletion operation. Use the back side of this page for answering this question.