**Essential Algorithms** **Name:** ____________________

**CISC 233 Spring 2025**

**Exam 01**

**03/11/2025**

**Time Limit: 90 Minutes** Mina Gabriel

**Short Time Questions** 25 - points

1. (5 points) Give an example of $O(n!)$ algorithm.

2. (5 points) What are the conditions that must be met for binary search to be applicable, and how does violating these conditions affect performance?

3. (5 points) Consider the function $f(n) = 5^{n+1} + 4^{n+2}$. Determine the function $g(n)$ that describes the Big-O notation for $f(n)$. Find appropriate constants $c > 0$ and $n_0 > 0$ such that:
$$f(n) \leq c \cdot g(n), \quad \text{for all } n \geq n_0.$$

4. (5 points) Considering that deletion in a Max-Heap is only performed at the root, what are the best and worst-case scenarios for removing the root element? **Why**?

5. (5 points) A Red-Black Tree is a self-balancing binary search tree that maintains balance through specific properties.
   (a) List and explain the key properties that define a Red-Black Tree. Why are these properties necessary?
   (b) Red-Black Trees are widely used in various applications. Provide a real-world scenario where using a Red-Black Tree is advantageous.

**Medium Time Questions** 30 - points

1. (10 points) Skip the class constructors, what is the upper-bound run-time of each line of the code listing below? (Don't Explain the code)

Listing 1: Search an Element in a Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node

    def search(self, key):
        current = self.head
        position = 0
        while current:
            if current.data == key:
                return position
            position += 1
        return f"Element {key} not found"

# Example Usage
ll = LinkedList()
ll.insert(10)
ll.insert(20)
ll.insert(30)
ll.insert(40)

print(ll.search(30))  # Should return position 2
print(ll.search(50))  # Should return not found
```

2. (10 points) Given the following list of numbers:

$$A = [15, 8, 20, 5, 14, 25, 18]$$

Perform the **Min-Heapify** operation to transform the list into a valid **Min-Heap**.

(a) Show the **step-by-step transformations** of the array after applying **heapify**.

(b) Draw the final **Min-Heap representation** in tree form.

3. (10 points) Consider the following Red-Black Tree before insertion:

```
      30 (B)
     /     \
   20 (B)   40 (B)
  /           \
10 (R)         50 (R)
```

Now, suppose we insert a new node 60 (R) into the tree as a right child of 50, potentially violating the Red-Black Tree properties.

(a) Identify which Red-Black properties are violated after the insertion.

(b) Perform the necessary recoloring and rotations to restore the Red-Black Tree properties. Draw the final balanced Red-Black Tree after fixing violations.

(c) Determine the height of the tree and discuss how Red-Black Trees maintain O(log n) height.

**Long Time Questions** 45 - points

1. (15 points) An algorithm runs in

$$T(n) = O(n^2 + n \log n + 5n + 100).$$

Surprisingly, the algorithm takes 180 seconds to complete when $n = 100$, but only 290 seconds when $n = 200$. Use direct calculation, identify which term in $T(n)$ is likely to influence the runtime the most, show all steps.

2. (15 points) The same array is given to the four sorting algorithms listed below

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 9 | 6 | 4 | 1 | 7 | 3 | 0 | 8 | 5 |

a. Heap Sort

b. Insertion Sort

c. Merge Sort

d. Selection Sort

The following are the intermediate results produced by each of them at a certain time during the sorting process. Your task is to label each array to indicate which algorithm produces it.

(____)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 9 | 0 | 3 | 5 | 7 | 8 |

(____)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 4 | 1 | 0 | 6 | 7 | 8 | 9 |

(____)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 7 | 9 | 3 | 0 | 8 | 5 |

(____)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 4 | 5 | 6 | 3 | 0 | 2 | 1 |

3. (15 points) You are given a dataset represented as a table with three columns: $A$, $B$, and $C$. Your task is to sort this dataset in alphabetical order, first by column $A$, then by column $B$, and finally by column $C$.

   (a) (8 points) Describe, either in pseudocode or in English, the process you would use to sort the table. Propose an optimal algorithm for addressing this multi-column sorting problem. Specify the time complexity of your solution using Big-O notation. Evaluate which sorting algorithm is best suited for this task, provide a clear justification for your selection.

   (b) (7 points) Once the table is sorted, how can you efficiently locate a row with a specific combination of values in columns $A$, $B$, and $C$? Develop a function (in pseudocode or detailed English) to perform this search within the sorted table, and specify the time complexity of your search approach using Big-O notation.

   *(Hint: Linear search is not the most efficient method.)*

## Example of the table before and after sorting:

**Unsorted Table:**

| Column A | Column B | Column C |
|---|---|---|
| C | A | B |
| A | C | A |
| B | B | C |
| A | A | B |
| C | C | A |
| B | A | C |
| A | B | A |
| C | B | C |
| B | C | B |
| A | C | C |

**Sorted Table by ($A$, $B$, then $C$):**

| Column A | Column B | Column C |
|---|---|---|
| A | A | B |
| A | B | A |
| A | C | A |
| A | C | C |
| B | A | C |
| B | B | C |
| B | C | B |
| C | A | B |
| C | B | C |
| C | C | A |