Student ID:
Name:
Signature:

COMP 203 DATA STRUCTURES AND ALGORITHMS

2024 Fall Semester Midterm

Date=19.11.2024 12:00 pm **Duration:** 2 Hours **Total=**100 points

Rules: 1. This is a closed book exam. No phone or note is allowed.

2. Read the questions carefully. Don't forget the sub parts of the question.

3. If the question says "explain", explain it clearly.

4. AGU cheating policy is applied during this exam.

5. Write the answers with readable handwriting and clear order.

Q1(20pt)	Q2(25pt)	Q3(25pt)	Q4(30pt)	Total(100pt)

QUESTIONS

1. Recursion and Time Complexity (2*10=20 points)

a. Write a recursive boolean method that checks if the given string is a palindrome or not. Palindrome means that a word, sentence that reads the same backward or forward.

Examples: Input: "radar" Output: true, Input: "Step on no pets" Output: true, Input: "Home" Output: false.

b. What is the average case time complexity of the function you implemented? (You may use Big-O notation) Explain why.

ANSWER:

a.
```
static boolean isPalRec(String str,
                        int s, int e)
  {
     // If there is only one character
     if (s == e)
        return true;

     // If first and last
     // characters do not match
     if ((str.charAt(s)) != (str.charAt(e)))
        return false;
```

Student ID:
Name:
Signature:

```
        // If there are more than
        // two characters, check if
        // middle substring is also
        // palindrome or not.
        if (s < e + 1)
            return isPalRec(str, s + 1, e - 1);

        return true;
    }

    static boolean isPalindrome(String str)
    {
        int n = str.length();

    // An empty string is
    // considered as palindrome
        if (n == 0)
            return true;
        return isPalRec(str, 0, n - 1);
    }
```

b. Time complexity is O(n) because we iterate half of the string which is n/2.

2. SLL, Time Complexity and Generics (5x5=25 points)

a. Write a pseudocode or java code for a Node<E> class with its constructors.

b. Write a pseudocode or java code for a SinglyLinkedList<E> class with its constructors.

c. Write a pseudocode or java code for a function with the name “void delete(E deletedValue)” to delete a node with the value *deletedValue* in a singly linked list. This will be implemented in SinglyLinkedList class.

Example: This is singly linked list:

A->B->C->D->null

After delete(C) it will be;

A->B->D->null

Student ID:
Name:
Signature:

d. What is the average time complexity of “void delete(E deletedValue)” with Big-O? Explain why.

e. Write a pseudocode or java code for a function with the name “boolean isEmpty()” to check if the given singly linked list is empty or not.

ANSWER:

a, b, c, e:

```
class Node<E>:
    E data
    Node<E> next

    // Constructors
    Node(data)
        this.data = data
        this.next = null

    Node(data, next)
        this.data = data
        this.next = next

class SinglyLinkedList<E>:
    Node<E> head
    // Constructors
    SinglyLinkedList()
        this.head = null

    SinglyLinkedList(head)
        this.head = head

    // Function to delete a node with a specific value
    void delete(E deletedValue)
        if head is null
            return  // List is empty, nothing to delete
```

Student ID:
Name:
Signature:

```
if head.data is equal to deletedValue
   head = head.next  // Delete the head node
   return

current = head
previous = null

while current is not null and current.data is not equal to deletedValue
   previous = current
   current = current.next

if current is null
   return  // Node with the specified value not found

// Delete the node
previous.next = current.next
```

```
Boolean isEmpty()
If(head==null)
        Return true
Else
        Return false
```

2.d. The average time complexity of the **delete** function in the SinglyLinkedList class is $O(n)$, where n is the number of elements in the linked list. In the worst case, the function may need to traverse the entire list to find the node with the specified value. Each iteration of the while loop takes constant time, and the worst-case scenario is when the node to be deleted is at the end of the list.

3.DLL, Time Complexity (5*5=25 points)

a. Write a pseudocode or java code for Node<Integer> class to implement node for a Doubly Linked list. Implement its constructors.

b. Write a pseudocode or java code for DoublyLinkedList <Integer> class with its constructors.

Student ID:
Name:
Signature:

c. Write a pseudocode for a function with name “Node<Integer> AddAfter(int first, int addedValue)” for Doubly Linked list that inserts a node having *addedValue* after the node having the value *first* and returns the head of the DLL. Assume you have:

Example: We have DLL head⇔ 2⇔4⇔5⇔8⇔tail

After AddAfter (4,7);

head⇔ 2⇔4⇔7⇔5⇔8⇔tail and returns the head node.

d. What is the average time complexity of “Node< Integer> AddAfter(int first, int addedValue)” with Big-O? Explain why.

e. Write a pseudocode or java code for a function with the name “boolean isInDLL(int element)” to check if the given DLL has the *element* or not. Hint: You may use this method in section d.

ANSWER:

3. a, b, c, e

```
class Node<Integer>:
   Integer data
   Node<Integer> next
   Node<Integer> prev

   // Constructors
   Node(data)
      this.data = data
      this.next = null
      this.prev = null

   Node(data, next, prev)
      this.data = data
      this.next = next
      this.prev = prev

class DoublyLinkedList<Integer>:
   Node<Integer> head
   Node<Integer> tail

   // Constructors
   DoublyLinkedList()
      this.head = null
      this.tail = null
```

Student ID:
Name:
Signature:

```
DoublyLinkedList(head, tail)
    this.head = head
    this.tail = tail
```

d. Node<Integer> AddAfter(int first, int addedValue)

```
// Check if the node with value 'first' exists in the list
if (!isInDLL(first)) {
    Output("Value " + first + " not found in the list.")
    return head // Return the head as no changes are made
}
Node<Integer> current = head
// Traverse the list to find the node with value 'first'
while (current != null) {
    if (current.data.equals(first)) {
        // Create a new node with addedValue
        Node<Integer> newNode = new Node<>(addedValue)

        // Set pointers for the new node
        newNode.prev = current;
        newNode.next = current.next;

        // Update the next node's prev pointer if it exists
        if (current.next != null)
            current.next.prev = newNode;

        // Update the current node's next pointer
        current.next = newNode

        // Update the tail if the new node is added at the end
        if (newNode.next == null)
            tail = newNode
```

Student ID:
Name:
Signature:

```
            return head // Return the head of the list

        current = current.next
    }
    return head
}
```

e. boolean isInDLL(int element)

```
    Node<Integer> current = head

    // Traverse the list to check for the element
    while (current != null) {
        if (current.data.equals(element)) {
            return true // Element found
        }
        current = current.next
    }

    // Element not found
    return false
}
```

3.d. The average time complexity of the AddAfter function is O(n), where n is the number of elements in the doubly linked list. In the worst case, the function may need to traverse the entire list to find the node with the specified value ("first"). Each iteration of the while loop takes constant time, and the worst-case scenario is when the node to be added after is at the end of the list.

4. Queue, Array and Time Complexity (6x5=30 points)

a. Write a psueudocode or java code for Queue<Integer> class from a fixed size array. It must include instance variables and constructors.

b. Write a psueudocode or java code “void Enqueue(int element)” function to insert the value *element* to the queue.

Student ID:
Name:
Signature:

c. Write a psueudocode or java code “int Dequeue()” function to delete the element from the queue and return the deleted value.

d. Write a psueudocode or java code **“**int Size()” function to return the size of the queue.

e. Write a psueudocode or java code “Boolean IsFull()” function to check if the queue is full or not.

f. What is the average time complexity of each function (Enqueue(int element), Dequeue(), Size(), IsFull())? Explain why. You may use Big-O notation.

ANSWER:

4.a,b,c,d,e

```
class Queue<Integer>:
    Integer[] array
    int front
    int rear
    int capacity
    int size

    // Constructor
    Queue(capacity)
        this.array = new Integer[capacity]
        this.front = 0
        this.rear = -1
        this.capacity = capacity
        this.size = 0
```

void Enqueue(int element):

```
    if IsFull()
        // Queue is full, throw an exception or handle accordingly
        return

    // Increment rear and add the element to the queue
```

```
    rear = (rear + 1) % capacity
    array[rear] = element
    size = size + 1
int Dequeue():
    if size equals 0
        // Queue is empty, throw an exception or handle accordingly
        return -1  // Assuming -1 is an invalid value for the queue
    // Retrieve the front element, increment front, and decrement size
    int deletedElement = array[front]
    front = (front + 1) % capacity
    size = size - 1
    return deletedElement
int Size():
    return size
Boolean IsFull():
    return size equals capacity
```

f.

1. **Enqueue(int element):** The average time complexity of the **Enqueue** function is *O*(1). This is because adding an element to the rear of the circular array takes constant time.
2. **Dequeue():** The average time complexity of the **Dequeue** function is *O*(1). Dequeueing from the front of the circular array also takes constant time.
3. **Size():** The **Size** function simply returns the size, so its time complexity is *O*(1).
4. **IsFull():** The **IsFull** function checks whether the size equals the capacity, which also takes constant time, resulting in a time complexity of *O*(1).

The average time complexity of each function is constant *O*(1)) because each operation involves a fixed number of array accesses or comparisons, regardless of the size of the queue.