Data Structures CSC 212 | Midterm Exam - Fall 2020

Date: 13/10/2020 | Duration: 90 minutes

Guidelines

- No calculators or any other electronic devices are allowed in this exam.

Student ID:

Section: | Name:

Instructor:

1	2	3	4	Total

Question 1 .... 15 points

Clearly mark *one* answer for each of the following:

1. For $f(n) = 2n^{\frac{7}{2}} + 6n^2 - n^3$, which of the following is *true*? $f(n)$ is:

   [✓ Ⓐ $O(n^{\frac{7}{2}})$] Ⓑ $O(n^3)$ Ⓒ $O(n^2)$ Ⓓ A and B Ⓔ None

2. For $f(n) = 14n + 2n\log n - 6n^{\frac{1}{2}}$, which is the *most appropriate*? $f(n)$ is:

   Ⓐ $O(n)$ Ⓑ $O(n^{1/2})$ Ⓒ $O(n^2)$ [✓ Ⓓ $O(n\log n)$] Ⓔ None

3. For $f(n) = \log(n+1)^2$, which is the *most appropriate*? $f(n)$ is:

   $\log(n+1)^2$ $2\log(n+1)$

   [✓ Ⓐ $O(\log n)$] Ⓑ $O((\log n)^2)$ Ⓒ $O(\log(n^2))$ Ⓓ $O(n^2)$ Ⓔ None

4. Suppose you had an algorithm $\mathcal{A}$. When it runs on a list l of size n, $\mathcal{A}$ calls algorithm $\mathcal{B}$ on each of the first three iterations. On the remaining iterations, it calls algorithm $\mathcal{C}$. Suppose $\mathcal{B}$ is $O(n)$ and $\mathcal{C}$ is $O(1)$. Which running time *best describes* algorithm $\mathcal{A}$:

   A → O(n) B . 3+C O(n) O(1)

   Ⓐ $O(n)$ [✓ Ⓑ $O(n^2)$] Ⓒ $O(2^n)$ Ⓓ $O(1)$ Ⓔ None

. Suppose that an algorithm $\mathcal{A}$ iterates over a list l of size n. For each element in the first half of l $\mathcal{A}$ executes some $O(\log n)$ operation. For elements in the second half of l, $\mathcal{A}$ executes some $O(1)$ operation. Which running time *best describes* algorithm $\mathcal{A}$?

Ⓐ $O(n)$ Ⓑ $O(n\log n)$ Ⓒ $O(n+\log n)$ Ⓓ $O(n^2\log n)$ Ⓔ None

For two functions $f(n)$ and $g(n)$, suppose $f(n)$ is $O(g(n))$. What is the best big-O for $f(n) + g(n)$?

[✓ Ⓐ $O(g(n))$] Ⓑ $O(2f(n))$ Ⓒ $O(f(n)\cdot g(n))$ Ⓓ $O(f(n)/g(n))$ Ⓔ None

Suppose $f(n)$ is $O(g_1(n))$ and $h(n)$ is $O(g_2(n))$. What is the best big-O of $f(n)\cdot h(n)$?

Ⓐ $O(g_1(n) + g_2(n))$ Ⓑ $O(g_1(n))$ Ⓒ $O(g_1(n)\cdot g_2(n))$ Ⓓ $O(g_2(n))$ Ⓔ None

Question 2 ....25 points

In a real estate management application, information about properties (e.g., homes, farms, and land) and their owners are stored in the two classes `Property` and `Owner` shown below. Notice that a property may have multiple owners, and an owner may participate in owning several properties.

```
public class Property {
  public String id;
  public String propertyName;
  public double value;
  public List<Owner> owners; // Owners of this property
  public Property(String id, String propertyName, List<Owner> owners) {
    this.id= id;
    this.propertyName= propertyName;
    this.owners= owners;}
}
```

```
public class Owner {
  public String firstName;
  public String lastName;
  public Owner(String firstName, String lastName) {
    this.firstName= firstName;
    this.lastName= lastName;}
}
```

Write the method `ownersOf` that takes as input a list of properties and a particular property's ID, and checks if that property is included in the list. The method should return a list containing all the owners of this property if it exists; otherwise it should return an empty list. The method signature is: `static List<Owner> ownersOf(List<Property> properties, String id)`.

1. Line 1:
   - [✓ Ⓐ `if (!properties.empty()){`]
   - Ⓑ `while (!properties.last()){`
   - Ⓒ `properties.findFirst();`
   - Ⓓ `if (properties.head != null){`
   - Ⓔ None
2. Line 2:
   - Ⓐ `if (!properties.empty()){`
   - Ⓑ `if (properties.retrieve().id.equals(id))`
   - Ⓒ `properties.current = head;`
   - [✓ Ⓓ `properties.findFirst();`]
   - Ⓔ None
3. Line 3:
   - Ⓐ `return properties.retrieve().owners;`
   - Ⓑ `while (properties.last()){`
   - [✓ Ⓒ `while (!properties.last()){`]
   - Ⓓ `while (properties.current.next != null){`
   - Ⓔ None
4. Line 4:
   - Ⓐ `properties.findNext(); }`
   - Ⓑ `if (properties.retrieve().id.equals(id))`
   - [✓ Ⓒ `if (properties.retrieve().id == id)`]
   - Ⓓ `if (properties.data.id.equals(id))`
   - Ⓔ None
5. Line 5:
   - Ⓐ `if (properties.retrieve().id.equals(id))`
   - Ⓑ `return properties.data.owners;`
   - Ⓒ `return properties.owners;`
   - [✓ Ⓓ `return properties.retrieve().owners;`]
   - Ⓔ None
6. Line 6:
   - [✓ Ⓐ `properties.findNext(); }`]
   - Ⓑ `properties.current = current.next; }`

Ⓒ `return properties.retrieve().owners; }`

Ⓓ `properties.findFirst(); }`

Ⓔ None

7. Line 7:

Ⓐ `if (!properties.retrieve().owners.empty())`

[✓ Ⓑ `if (properties.retrieve().id == id)`]

Ⓒ `if (properties.data.id.equals(id))`

Ⓓ `if (properties.retrieve().id.equals(id))`

Ⓔ None

8. Line 8:

Ⓐ `properties.retrieve().owners.remove(); }`

[✓ Ⓑ `return properties.retrieve().owners; }`]

Ⓒ `return properties.owners; }`

Ⓓ `return properties.data.owners; }`

Ⓔ None

9. Line 9:

Ⓐ `return new LinkedList<Property>();`

Ⓑ `return new LinkedList<Owner>();`

Ⓒ `return properties;`

[✓ Ⓓ `return null;`]

Ⓔ None

Question 3 ....25 points

Clearly mark *one* answer for each of the following:

1. Which of the following is true about linked implementation of queue?

[✓ Ⓐ In enqueue operation, if new nodes are inserted at the beginning, then in serve operation, nodes must be removed from the end.] [✓ Ⓑ In enqueue operation, if new nodes are inserted at the end, then in serve operation, nodes must be removed from the beginning.] [✓ Ⓒ Both of the above.] Ⓓ None of the above.

2. What does the method `f` below do ?

```
public static <T> void f(Queue<T> q) {
  if (q.length() > 0) {
    T e = q.serve();
    f(q);
    q.enqueue(e);
  }
}
```

① e→1 ② e→2 ③ e→3

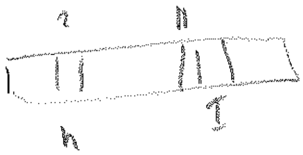
1 2 3 4 5 ④ e→4 ⑤ e→5

5→4

Ⓐ Leaves `q` unchanged. Ⓑ Empties `q`. Ⓒ Deletes the first element of `q` and inserts it at the end keeping the other elements in the same order. [✓ Ⓓ Reverses `q`.] Ⓔ None

3. Suppose we have a circular array implementation of the queue, with ten items in the queue stored at `data[2]` through `data[11]`. The capacity (that is `maxSize`) is 42. Where does the enqueue method place the new entry in the array?

Ⓐ data[1] Ⓑ data[11] [✓ Ⓒ data[12]] Ⓓ data[0] Ⓔ None



4. The function `g` below is member of `ArrayQueue`. What does it do?

```
public T g() {
  if(size == 0)
    return null;
  else {
    T e = data[head];
    return e;
  }
}
```

Ⓐ Return the front element. Ⓑ Enqueue. Ⓒ Serve. Ⓓ Return the last element.
Ⓔ None.

5. In the linked implementation of a queue, which of the pointers `head` and `tail` will change during an enqueue into a non-empty queue?

Ⓐ Only `head`. Ⓑ Only `tail`. Ⓒ Both `head` and `tail`. Ⓓ Depends on the size of the queue.
Ⓔ None.

6. What is the content of q at the end of the following code:

```
Queue<Integer> q = new LinkedQueue<Integer>()
q.enqueue(5);
q.serve();
q.enqueue(3);
q.enqueue(2);
q.enqueue(4);
q.serve();
q.serve();
q.enqueue(2);
```

Ⓐ 5, 3, 2 Ⓑ 5, 3, 2, 4 Ⓒ 4, 2, 3 Ⓓ 2, 4, 2, 3 Ⓔ None

Question 4 .... 35 points

We want to write a linked implementation of the ADT `UQueue` which is a linear structure that stores elements without repetition and allows to serve from both ends.

```
public interface UQueue<T> {
  int length();
  boolean full();
  // Insert e at the end if it does not already exist and return true, otherwise return false.
  boolean enqueue(T e);
  // Remove and return the first element (the oldest)
  T serveFirst();
  // Remove and return the last element (the newest)
  T servLast();
}
```

Complete the class `LinkedUQueue` below.

```
class Node<T> {
  public T data;
  public Node<T> next, prev;
  public Node(T data) {
    this.data = data;
    prev = next = null;
  }
}
public class LinkedUQueue<T> implements UQueue<T> {
  private Node<T> head, tail;
  private int size;
  public LinkedUQueue() {
    tail = head = null;
    size = 0;
  }
  public int length() {
    return size;
}
public boolean full() {
  return false;
}
```

1. Method enqueue.

```
1  public boolean enqueue(T e) {
2    if (size == 0) {
3      ...
4    } else {
5      Node<T> p = ...
6      while (...)
7        ...
8      if (...)
9        ...
10     ...
11     ...
12     ...
13   }
14   ...
15   ...
16 }
```

- Line 3:
  - (A) `tail = new Node<T>(e);`
  - (B) `tail = head = new Node<T>(e);`
  - (C) `tail = head = null;`
  - (D) `head = new Node<T>(e);`
  - (E) None
- Line 5:
  - (A) `Node<T> p = head;`
  - (B) `Node<T> p = null;`
  - (C) `Node<T> p = tail.prev;`
  - (D) `Node<T> p = head.next;`
  - (E) None
- Line 6:
  - (A) `while (p.next != null && !e.equals(p.data)`
    `)`
  - (B) `while (p != tail && !e.equals(p.data))`
  - (C) `while (e.equals(p.data))`
  - (D) `while (p != null && !e.equals(p.data))`
  - (E) None
- Line 7:
  - (A) `p = head.next;`
  - (B) `p.next = p;`
  - (C) `p = p.next;`
  - (D) `p = p.prev.next;`
  - (E) None
- Line 8:
  - (A) `if (p == e)`
  - (B) `if (p.equals(e))`
  - (C) `if (p == head)`
  - (D) `if (p != null)`
  - (E) None
- Line 9:
  - (A) `return p != tail;`
  - (B) `return false;`
  - (C) `return true;`
  - (D) `return p != null;`
  - (E) None
- Line 10:
  - (A) `tail.next = new Node<T>(e);`
  - (B) `tail.prev = new Node<T>(e);`
  - (C) `tail = new Node<T>(e);`
  - (D) `head.next = new Node<T>(e);`
  - (E) None
- Line 11:
  - (A) `tail.prev.next = tail;`
  - (B) `head.prev.next = tail;`
  - (C) `tail.next = tail;`
  - (D) `tail.next.prev = tail;`
  - (E) None
- Line 12:
  - (A) `head = head.prev;`
  - (B) `head.next.prev = tail;`

Ⓒ `tail = tail.next;`

Ⓓ `tail = tail.prev;`

Ⓔ None

- Line 14:

  Ⓐ `if (head.next != null)size++;`

  Ⓑ `size++;`

  Ⓒ `size--;`

  Ⓓ `if (tail.prev != null)size++;`

  Ⓔ None

- Line 15:

  Ⓐ `return head.next != null;`

  Ⓑ `return e != null;`

  Ⓒ `return true;`

  Ⓓ `return e;`

  Ⓔ None

2. Method `serveFist`.

```
1  public T serveFirst() {
2    T e = ...
3     ...
4     if (...)
5       ...
6     else
7       ...
8     ...
9     ...
10 }
```

- Line 2:

  Ⓐ `T e = new Node<T>(head.data);`

  Ⓑ `T e = tail.data;`

  Ⓒ `T e = head;`

  Ⓓ `T e = head.data;`

  Ⓔ None

- Line 3:

  Ⓐ `tail = tail.next;`

  Ⓑ `head = head.prev;`

  Ⓒ `head = head.next;`

  Ⓓ `tail = tail.prev;`

  Ⓔ None

- Line 4:

  Ⓐ `if (tail == head)`

  Ⓑ `if (tail == null)`

  Ⓒ `if (head == null)`

  Ⓓ `if (size == 0)`

  Ⓔ None

- Line 5:

  Ⓐ `head.prev = null;`

  Ⓑ `head = null;`

  Ⓒ `tail = null;`

  Ⓓ `head = tail;`

  Ⓔ None

- Line 7:

  Ⓐ `head.next = null;`

  Ⓑ `tail.next = null;`

  Ⓒ `tail.prev = null;`

  Ⓓ `head.prev = null;`

  Ⓔ None

- Line 8:

  Ⓐ `size = 0;`

  Ⓑ `size--;`

  Ⓒ `size++;`

  Ⓓ `if (head !=null)size--;`

  Ⓔ None

- Line 9:

  Ⓐ `return head.data;`

  Ⓑ `return tail.data;`

  Ⓒ `return e.data;`

  Ⓓ `return e;`

  Ⓔ None

Method serveLast.

```
public T serveLast() {
  T e = ...
  ...
  if (...)
    ...
  else
    ...
  ...
  ...
}
```

- Line 2:

  Ⓐ `T e = head.data;`

  Ⓑ `T e = new Node<T>(head.data);`

  Ⓒ `T e = tail.data;`

  Ⓓ `T e = head;`

  Ⓔ None

- Line 3:

  Ⓐ `head = head.prev;`

  Ⓑ `tail = tail.next;`

  Ⓒ `head = head.next;`

  Ⓓ `tail = tail.prev;`

  Ⓔ None

- Line 4:

  Ⓐ `if (tail == head)`

  Ⓑ `if (size == 0)`

  Ⓒ `if (head == null)`

  Ⓓ `if (tail == null)`

  Ⓔ None

- Line 5:

  Ⓐ `head = tail;`

  Ⓑ `head.prev = null;`

  Ⓒ `head = null;`

  Ⓓ `tail = null;`

  Ⓔ None

- Line 7:

  Ⓐ `tail.next = null;`

  Ⓑ `tail.prev = null;`

  Ⓒ `head.prev = null;`

  Ⓓ `head.next = null;`

  Ⓔ None

- Line 8:

  Ⓐ `size++;`

  Ⓑ `size--;`

  Ⓒ `if (head !=null)size--;`

  Ⓓ `size = 0;`

  Ⓔ None

- Line 9:

  Ⓐ `return head.data;`

  Ⓑ `return e.data;`

  Ⓒ `return e;`

  Ⓓ `return tail.data;`

  Ⓔ None

Problem 4

1. Write the method *checkListEndsSymmetry* that receives a double linked list and an integer number k. The method checks if the double linked list has identical k elements going forward from the first element and backwards from the last one. The method returns *true* if they are identical, and *false* otherwise. The method signature is:

   public <T>boolean checkListEndsSymmetry(DoubleLinkedList<T>dl, int k)

   Example 4.1. *If* $dl = A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow B \leftrightarrow A$ *and* $k = 2$*, then the method should return* true*. If* $k = 3$*, it should return* false*, since* C *does not equal* D*.*

2. Write the method *bubbleSort* that sorts a double linked list of integers given as input using bubble sort. The method signature is:

   public void bubbleSort(DoubleLinkedList<Integer>l).

. . . .

Problem 5

1. Write the **recursive** method *reverse*, member of the class *LinkedList* that reverses the content of the list.

2. Write the **recursive** method *reverse*, member of the class *DoubleLinkedList* that reverses the content of the list.

. . . .