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: Name:

Section: 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:
   Ⓐ $O(\log n)$ Ⓑ $O((\log n)^2)$ Ⓒ $O(\log(n^2))$ Ⓓ $O(n^2)$ Ⓔ 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 $\mathcal{A}$:
   Ⓐ $O(n)$ Ⓑ $O(n^2)$ Ⓒ $O(2^n)$ Ⓓ $O(1)$ Ⓔ None
5. 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
6. 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
7. 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:
   - (A) `if (!properties.empty()){`
   - (B) `while (!properties.last()){`
   - (C) `properties.findFirst();`
   - (D) `if (properties.head != null){`
   - (E) None
2. Line 2:
   - (A) `if (!properties.empty()){`
   - (B) `if (properties.retrieve().id.equals(id))`
   - (C) `properties.current = head;`
   - (D) `properties.findFirst();`
   - (E) None
3. Line 3:
   - (A) `return properties.retrieve().owners;`
   - (B) `while (properties.last()){`
   - (C) `while (!properties.last()){`
   - (D) `while (properties.current.next != null){`
   - (E) None
4. Line 4:
   - (A) `properties.findNext(); }`
   - (B) `if (properties.retrieve().id.equals(id))`
   - (C) `if (properties.retrieve().id == id)`
   - (D) `if (properties.data.id.equals(id))`
   - (E) None
5. Line 5:
   - (A) `if (properties.retrieve().id.equals(id))`
   - (B) `return properties.data.owners;`
   - (C) `return properties.owners;`
   - (D) `return properties.retrieve().owners;`
   - (E) None
6. Line 6:
   - (A) `properties.findNext(); }`
   - (B) `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);
  }
}
```

Ⓐ 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;`

(C) `tail = tail.next;`

(D) `tail = tail.prev;`

(E) None

- Line 14:

(A) `if (head.next != null)size++;`

(B) `size++;`

(C) `size--;`

(D) `if (tail.prev != null)size++;`

(E) None

- Line 15:

(A) `return head.next != null;`

(B) `return e != null;`

(C) `return true;`

(D) `return e;`

(E) None

2. Method `serveFist`.

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

- Line 2:

(A) `T e = new Node<T>(head.data);`

(B) `T e = tail.data;`

(C) `T e = head;`

(D) `T e = head.data;`

(E) None

- Line 3:

(A) `tail = tail.next;`

(B) `head = head.prev;`

(C) `head = head.next;`

(D) `tail = tail.prev;`

(E) None

- Line 4:

(A) `if (tail == head)`

(B) `if (tail == null)`

(C) `if (head == null)`

(D) `if (size == 0)`

(E) None

- Line 5:

(A) `head.prev = null;`

(B) `head = null;`

(C) `tail = null;`

(D) `head = tail;`

(E) None

- Line 7:

(A) `head.next = null;`

(B) `tail.next = null;`

(C) `tail.prev = null;`

(D) `head.prev = null;`

(E) None

- Line 8:

(A) `size = 0;`

(B) `size--;`

(C) `size++;`

(D) `if (head !=null)size--;`

(E) None

- Line 9:

(A) `return head.data;`

(B) `return tail.data;`

(C) `return e.data;`

(D) `return e;`

(E) None

Method serveLast.

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

- Line 2:
  - (A) `T e = head.data;`
  - (B) `T e = new Node<T>(head.data);`
  - (C) `T e = tail.data;`
  - (D) `T e = head;`
  - (E) None
- Line 3:
  - (A) `head = head.prev;`
  - (B) `tail = tail.next;`
  - (C) `head = head.next;`
  - (D) `tail = tail.prev;`
  - (E) None
- Line 4:
  - (A) `if (tail == head)`
  - (B) `if (size == 0)`
  - (C) `if (head == null)`
  - (D) `if (tail == null)`
  - (E) None
- Line 5:
  - (A) `head = tail;`
  - (B) `head.prev = null;`
  - (C) `head = null;`
  - (D) `tail = null;`
  - (E) None
- Line 7:
  - (A) `tail.next = null;`
  - (B) `tail.prev = null;`
  - (C) `head.prev = null;`
  - (D) `head.next = null;`
  - (E) None
- Line 8:
  - (A) `size++;`
  - (B) `size--;`
  - (C) `if (head !=null)size--;`
  - (D) `size = 0;`
  - (E) None
- Line 9:
  - (A) `return head.data;`
  - (B) `return e.data;`
  - (C) `return e;`
  - (D) `return tail.data;`
  - (E) None