THE UNIVERSITY OF WESTERN AUSTRALIA

MID SEMESTER EXAMINATION
April 2016

SCHOOL OF COMPUTER SCIENCE & SOFTWARE ENGINEERING

DATA STRUCTURES AND ALGORITHMS CITS2200

This Paper Contains:
6 Pages
10 Questions

Time allowed : **Forty five minutes**

Marks for this paper total 10.
Candidates should answer **ALL** Questions.

Q1. $f(x) = O(\log^2 n)$, $g(x) = O(n \log n)$ and $h(x) = O(n)$. Which of the following statements is true?

(A) $g(x) = O(f(x))$
(B) $h(x) = O(f(x))$
(C) $g(x) = O(h(x))$
(D) $f(x) = O(g(x))$

Q2. The time complexity of the `merge` method in the `Merge Sort` algorithm is:

(A) $O(\log n)$
(B) $O(1)$
(C) $O(n)$
(D) none of the above.

Q3. The time complexity of the `Partition` method in the `Quick Sort` algorithm is:

(A) $O(n)$
(B) $O(n^2)$
(C) $O(\log n)$
(D) constant time.

Q4. The following is the code for the `dequeue()` method for the recursive or linked implementation of a Queue:

```
public Object dequeue () throws Underflow{
if (!isEmpty()){
   Object o = first.item;
   <missing line 1.>
   if (isEmpty())
      <missing line 2.>
   return o;
  }
 else throw new Underflow("dequeuing from empty queue");
}
```

The missing lines are:

(A) `1.first = first.successor; 2.last = last.successor;`
(B) `1.first = first.successor; 2.last = null;`
(C) `1.first = null; 2.last = null;`
(D) `1.first.successor = first; 2.last = null;`

Q5. The following is the code for `previous` method in a singly linked list. It shifts the position of the `window` one position to the left, i.e., previous to the current position:

```
public void previous (WindowLinked w) throws
OutOfBounds {
   if (!isBeforeFirst(w)) {
      Link current = before.successor;
      Link previous = before;
         while (current != w.link) {
            <missing line 1.>
            current = current.successor;
         }
         <missing line 2.>;
       }
       else throw new OutOfBounds ("Calling previous before start of list.");
     }
```

The missing lines are:

(A) `1. current=previous; 2. w.link = previous;`
(B) `1. previous = current; 2. w.link = current;`
(C) `1. previous = current; 2. w.link = previous;`
(D) `1. current = previous; 2. w.link = current;`

Q6. We want to add an extra method called `multiDequeue` to the implementation of a *Queue*. *multiDequeue(n)* removes n elements from the front of a `Queue`. If we perform n `multiDequeue` operations, the amortized cost is:

(A) $O(n)$;
(B) $O(n^2)$;
(C) $O(n^3)$;
(D) $O(1)$;

Q7. The number of edges in a tree with n nodes is:

(A) $n - 1$;
(B) n;
(C) $n \log n$;
(D) $\frac{n}{2}$;

Q8. We have a tree with n nodes. Which of the following statements about its height cannot be true?

(A) The height is $O(\log n)$;
(B) The height is $\frac{n}{2}$;
(C) The height is $O(n \log n)$;
(D) The height is 6;

Q9. Which of the following statements is incorrect?

(A) The worst-case complexity of insertion sort is $O(n^2)$;
(B) The worst-case complexity of merge sort is $O(n \log n)$;
(C) The worst-case complexity of quick sort is $O(n \log n)$;
(D) The worst-case complexity of quick sort is $O(n^2)$;

Q10. We want to implement a method called `popeye(i)` for the `stack` data structure. Given a stack `stack1` and an integer i, `popeye(i)` pops the i-th item from the top of `stack1` and keeps `stack1` otherwise as it was before. In other words, the only difference in the state of `stack1` before and after the `popeye(i)` operation is that the ith item from the top will be missing. We will assume that underflow will not occur during a `popeye(i)` operation. We propose two strategies for implementing `popeye(i)`.

1. We use an additional queue called `queue1`. We pop the first $i-1$ items from the top of `stack1` one by one and enqueue those items in `queue1` as we pop each item from `stack1`. We then pop the ith item from `stack1` and return it as a result of the execution of the `popeye(i)` method. We next dequeue each item from `queue1` and push it onto `stack1` until `queue1` is empty.

2. We use an additional stack called `stack2`. We pop the first $i-1$ items from the top of `stack1` one by one and push those items into `stack2` as we pop each item from `stack1`. We then pop the ith item from `stack1` and return it as a result of the execution of the `popeye(i)` method. We next pop each item from `stack2` and push it onto `stack1`, until `stack2` is empty.

Which of the following statements is true?

(A) Both strategies correctly implement the `popeye(i)` method.
(B) Only the first strategy correctly implements the `popeye(i)` method.
(C) None of the strategies correctly implements the `popeye(i)` method.
(D) Only the second strategy correctly implements the `popeye(i)` method.

END OF PAPER