CS 284: Mid-Term (50 Minutes) – Fall 2018 – Topic 1

October 12, 2018

Student Name:

Honor Pledge:

Grade sheet:

Problem 1 (20 points)	
Problem 2 (20 points)	
Problem 3 (20 points)	
Problem 4 (20 points)	
Problem 5 (20 points)	

Problems

Problem 1. Indicate for each of the following code fragments its complexity using big-$\mathcal{O}$ notation. You may assume that $n > 2$. In case you might require it, here is the formula for the sum of the numbers from 1 to n: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. You must justify your answer by proving a polynomial $T(n)$ that approximates the execution time.

```
// a
for(int i=0; i<n; i++) {
  for(int j=n; j>i+2; j--) {
    System.out.println(i + " " + j);
  }
}

// b
for(int i=1; i<n; i=i*2) {
    for(int j=0; j<n; j++) {
         System.out.println(i + " " + j);
    }
  }
}
```

Problem 2. Consider the class `SingleLLP` which is just like the class `SingleLL` seen in class except that nodes also have a *priority*. In other words, the inner class `Node` of `SingleLL` is replaced by the following one:

```
private static class NodeP<E>{

        private E data;
        private int priority; // new data field
        private NodeP<E> next;

        public NodeP(E data, int priority, NodeP<E> next) {
            this.data = data;
            this.priority = priority;
            this.next = next;
        }

        public NodeP(E data, int priority) {
            this(data, priority, null);
        }
    }
```

Implement the following operation as part of the `SingleLLP` class described below::

```
public int top_priority()
```

that returns the priority of the node in the list that has the highest priority. If the list is empty, it should throw an `EmptyListException`. Note that the list is not sorted in any way.

Problem 3. Implement the following new operation of the class `SingleLLP` :

```
public void bump_if(int lower_bound)
```

that increments in one unit the priority of every node whose current priority is `lower_bound` or more. Note that the list is not sorted in any way. What is the time complexity of your implementation?

This page is intentionally left blank. You may use it for writing your answers.

Problem 4. Suppose that you are given a class that implements a Queue and you execute the following lines of code:

```
public static void main(String[] args){
  Queue<Integer> myQueue = new LinkedList<Integer>();
  myQueue.offer(3);
  myQueue.offer(9);
  myQueue.offer(7);
  x = myQueue.poll();
  y = myQueue.remove();
  myQueue.offer(x+y);
  z = myQueue.remove();
  w = myQueue.poll();
}
```

What are the values of:

- $x =$
- $y =$
- $z =$
- $w =$

Problem 5. Suppose you want to access the items in a linked-list of nodes with priority based on their *priority*, rather than the order in which they were inserted in the list. It would be convenient to have a *priority index* so that access to the node with any given priority takes constant time. A priority index is an array `priority_index` of type `NodeP<E>[]` such that `priority_index[i]` is a reference to the node of priority `i`. If there is no node of that priority, it simply holds the value `null`. Implement the operation:

```
private NodeP<E> build_priority_index()
```

that builds such a priority index. You may assume that no two nodes have the same priority.

This page is intentionally left blank. You may use it for writing your answers.