Western University
Department of Computer Science

CS1027b Foundations of Computer Science II
Final Exam
April 29, 2022

PART I	
PART II	
20	
21	
22	
23	
24	
25	
26	
27	
Total	

Last Name: Solution

First Name: ____________________

Student Number: ____________________

Section Number (1 - Sarlo, 2 - Morey, 3 - Solis-Oba): ____________

Instructions

- Fill in your name, student number, and section number.
- The exam is 3 hours long.
- The exam has 11 pages and 27 questions, including 8 bonus marks.
- The first part of the exam consist of multiple choice questions. For each question circle **ONLY ONE** answer.
- For the second part of the exam, answer each question **ONLY** in the space provided.
- When you are done, raise your hand and one of the TA's will collect your exam.

Part I. Multiple Choice Questions

For each multiple choice question circle **ONLY ONE** answer.

1. (1 mark) In Assignment 3, you were asked to implement class DLStack.java with methods pop() and pop(int k). This is an example of ______________.

   (A) Overriding [✓ (B) Overloading] (C) Dynamic binding (D) Polymorphism

2. (1 mark) Arrays are allocated memory in the execution stack.

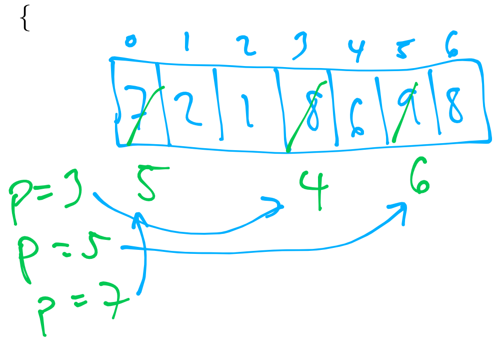
   (A) True [✓ (B) False]

3. (1 mark) There can be only one static object in any Java program.

   (A) True [✓ (B) False]

4. (3 marks) Consider the following code fragment.

```
public static void mystery (int[] arr, int p) {
   if (p >= arr.length) {
       arr[0] = 5;
       return;
   }
   int q = (arr.length + p) % arr.length;
   arr[q] = p+1;
   mystery(arr, p+2);
}
public static void main(String[] args) {
   int[] a = {7, 2, 1, 8, 3, 9, 4};
   mystery(a, 3);
}
```



   What are the values stored in array a and the end of the execution of method main?

   (A) {5, 2, 1, 8, 3, 9, 4} (B) {5, 2, 4, 8, 6, 9, 8} [✓ (C) {5, 2, 1, 4, 3, 6, 4}] (D) {5, 2, 1, 4, 3, 9, 4}

5. (2 marks) Consider the following code fragment:

```
int i = 0; int[] arr = new int[5]; arr[i++] = 3;
```

   Which of the following statement is true after executing this code?

   [✓ (A) `arr[0] = 3, i = 1`] (B) `arr[1] = 3 , i = 1` (C) `arr[0] = 3, i = 0`
   (D) `arr[1] = 3, i = 0`

6. (3 marks) Consider the following Java code.

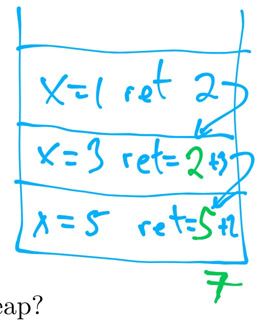
```
public class MyClass {
   private static int size = 10;
   private static int[] a;
   public MyClass(int capacity) {
       size = capacity;
       a = new int[size];
   }
   public static void main(String[] args) {
       MyClass c = new MyClass(4);
       MyClass b = new MyClass(8);
       System.out.println(c.size+","+b.size);
   }
}
```

   What is printed when the main method is executed?

   (A) 4,4 (B) 4,8 [✓ (C) 8,8] (D) 10,10 (E) 8,10

7. (2 marks) Consider the following method:

```
public int m(int x) {
   if (x == 1) return 2;
   else if (x > 4) return m(x - 2) + 2;
   else return m(x - 2) + x;
}
```



What value does `m(5)` return?

(A) 2   (B) 4   (C) 5   [✓ (D) 7]   (E) 9

8. (2 marks) Which of the following arrays represents a min heap?

A1						
	3	5	7	4	6	9

A2						
	5	6	8	7	9	7

A3						
	4	7	6	8	9	8

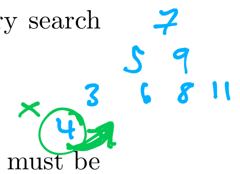
✓

(A) A1   (B) A2   [✓ (C) A3]   (D) None   (E) All of them

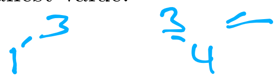
9. (2 marks) The following array represents a complete binary tree. Does it represent a binary search tree?

(A) Yes   [✓ (B) No]

7	5	9	3	6	8	11	4



10. (2 marks) In a binary search tree with at least 2 nodes, the second smallest value in the tree must be stored in the parent of the node storing the smallest value.



(A) true   [✓ (B) False]

11. (3 marks) Consider the following Java classes

```
public class A {
   public int value = 3;
}
public class B extends A {
   public int value;
   public B(int i) {
      value = i;
   }
}
```

What does the following code print?

```
A var1 = new B(4);
B var2 = new B(7);                  3           7
System.out.println(var1.value+var2.value);
```

(A) 4   (B) 6   (C) 7   [✓ (D) 10]   (E) 11

12. (3 marks) Consider the following Java code.

```
public class A {
   private String s;
   public void m(String str) {
      str = new String("hello");
   }
}
```

Which of the following statements is correct?

(A) The String object referenced by `str` is allocated memory in the heap, while variables `s` and `str` are allocated memory in the execution stack.
[✓ (B)] Variable `s` and the string object referenced by `str` are allocated memory in the heap, but variable `str` is allocated memory in the execution stack.
(C) The String object referenced by `str`, the variables `str`, and `s` are allocated memory in the heap.
(D) Variables `s` and `str` are allocated memory in the heap; the String object referenced by `str` is allocated memory in the execution stack.

13. (4 marks) Consider a **non-empty** binary tree in which every node stores an integer value. We wish to design an algorithm that given the root `r` of the tree and a value `k` it returns `true` if the value `k` **IS NOT** stored in the tree and it returns false otherwise. Which of the following algorithms correctly solves this problem?

[✓ (A)] A1 (B) A2 (C) A3 (D) A1 and A2 (E) A1 and A3 (F) None of them

```
public boolean A1(BinaryTreeNode r, int k) {
   if (r.getValue() == k) return false;
   else {
       if (r.leftChild() != null)
           if (A1(r.leftChild(),k) == false) return false;
       if (r.rightChild() != null)
           return A1(r.rightChild(),k);
       return true;
   }
}
```

```
public boolean A2(BinaryTreeNode r, int k) {
   boolean tmp = true;
   if (r.getValue() == k) tmp = false;
   else {
       if (r.leftChild() != null) tmp = A2(r.leftChild(),k);
       if (r.rightChild() != null) tmp = A2(r.rightChild(),k);
   }
   return tmp;
}
```

$$A2\left(\begin{tikzcd}[column sep=small] & 2 \arrow[dl, no head] \arrow[dr, no head] & \\ 1 & & 2\end{tikzcd}, 1\right) = True!$$

```
public boolean A3(BinaryTreeNode r, int k) {
   if (r.getValue() != k) return true;
   else {
       if (r.leftChild() != null)
           if (A3(r.leftChild(),k) == true) return true;
       if (r.rightChild() != null) return A3(r.rightChild(),k);
       return false;
   }
}
```

$$A3\left(\begin{tikzcd}[column sep=small] & 2 \arrow[dl, no head] \arrow[dr, no head] & \\ 1 & & 1\end{tikzcd}, 1\right) = true$$

14. (3 marks) Consider the following code.

```
public void sort(int[] a, int n) {
   for(int i = 0; i < n - 1; i = i + 1) {
       int j = i;
       int tmp = a[i+1];
       (**)
       a[j+1] = tmp;
   }
}
```

Which code must be inserted at the point marked (**) to sort the array in decreasing order? Array `a` stores `n` **different integer values**.

(A) `while (j >=0 && tmp < a[j]) a[j+1] = a[j--];`
[✓ (B)] `while (j >=0 && tmp > a[j]) a[j+1] = a[j--];`
(C) `while (j >= 0 && tmp < a[j]) a[j-1] = a[j++];`
(D) `while (j >= 0 && tmp > a[j]) a[j+1] = a[j++];`

15. (3 marks) Consider the following Java code.

```
private static void m(int[] a) {
   try {
       if (a.length == 0) System.out.print("Empty array.  ");
       else System.out.print("Non-empty array.  ");
   }
   catch (NullPointerException e) {
       System.out.print("Error 1.  ");
   }
   catch (IndexOutOfBoundsException e1) {
       System.out.println("Error 2.  ");
   }
}
public static void main (String[] args) {
   int[] a = null;
   try {
       m(a);
   }
   catch (Exception e) {
       System.out.print("Error 3.  ");
   }
}
```

When the `main` method is executed what message(s) will be printed?

[✓ (A) Error 1.] (B) Error 2. (C) Error 3. (D) Empty array. (E) Non-empty array. (F) Empty Array. Error 1.

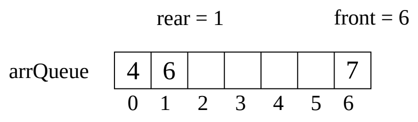
16. (3 marks) Consider the following implementation of method `remove(LinkedNode target)` that removes the node pointed by `target` from a doubly linked list. Instance variable `front` points to the first node of the list and instance variable `rear` points to the last node of the list. Methods `next()` and `prev()` return the next and previous nodes in the list, respectively, while `setNext()` and `setPrev()` change the pointers to the next and previous nodes.

```
public void remove(LinkedNode target) {
   if (rear == target) {
       rear = rear.prev();
       rear.setNext(null);  // what if rear now is null ie. front=rear
   }
   else {
       LinkedNode p = target.prev();
       p.setNext(target.next());
       (target.next()).setPrev(p);
       if (front == target) front = p;
}
```

Assume that `target` is a node of the list. Which of the following statements is true?

(A) The above code correctly removes `target` from the list.
[✓ (B) The above method will sometimes throw an exception.]
(C) The above method will never throw an exception, but it will sometimes not correctly remove `target` from the list.

17. (2 marks) Consider an empty stack `s` and a queue `q` storing the following values: 7, 4, and 6. The queue is implemented using a circular array `arrQueue` where variable `front` is the index of the first value and `rear` is the index of the last value in the queue, as shown in the following figure.



Consider the following code fragment

```
int i;
for (i = 0; i < 3; i = i+1) s.push(i);
for (i = 0; i < 3; i = i + 1) q.enqueue(s.pop());
for (i = 0; i < 2; i = i + 1) s.push(q.dequeue());
```

3 enqueue   rear=4
2. dequeue   front=1

What are the values of `front` and `rear` after the above code fragment is executed?

(A) `front = 2, rear = 4`   (B) `front = 1, rear = 3`   (C) `front = 2, rear = 0`
(D) `front = 0 rear = 5`   [✓ (E)] `front = 1, rear = 4`

18. (3 marks) Consider an implementation of a queue using a circular array `arrQueue` where variable `front` is the index of the first value and `rear` is the index of the last value in the queue, as shown in the figure of the previous question. Variable `count` indicates the number of values stored in the queue. Method `expandCapacity()` doubles the size of array `arrQueue`. Consider the following code for performing the enqueue operation:

```
public void enqueue(T value) {
   if (count++ == arrQueue.length) expandCapacity();
   rear = rear + 1;
   arrQueue[rear % arrQueue.length] = value;
   if (rear == arrQueue.length) rear = 0;
}
```

Which of the following statement is correct?

[✓ (A)] The above is a correct implementation of the `enqueue` operation.
(B) The above code will sometime throw an index out of bounds exception.
(C) The above code has compilation errors, so it cannot be executed.
(D) The above code will sometimes overwrite some of the values stored in the queue.

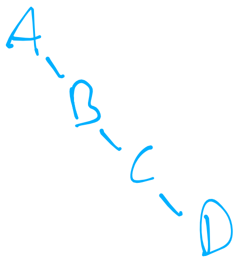
19. (1 mark) Insertion sort ✗, selection sort ✗, quicksort, and mergesort are examples of algorithms that are designed using the divide-and-conquer paradigm.

(A) True   [✓ (B)] False.

Part II. Written Answers

Write your answers **ONLY** in the spaces provided.

20. (6 marks) Draw a binary tree with 4 nodes, in which each node stores one of the values A, B, C, and D such that the preorder, inorder, and level-order traversals all visit the tree's nodes in the order A, B, C, D.



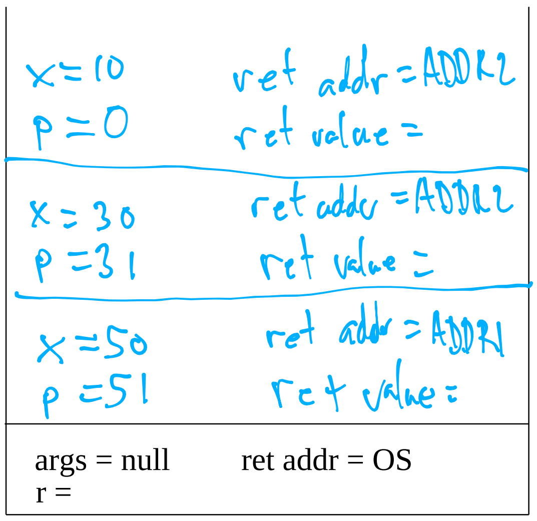
21. Consider the following code fragment.

```
public static int m1 (int x) {
   int p = x - 10;
   if (x < 15)
       return p; // * Pause execution here
   else {
       p = x + 1;
       return p + m1(x-20); // ADDR2
   }
}
public static void main (String[] args) {
   int r = m1(50); // ADDR1
   System.out.println(r);
}
```

Show the execution stack with all its activation records right BEFORE the underlined statement marked with an asterisk is executed (i.e. imagine that the above code is executed and paused JUST BEFORE the marked statement is executed; show the content of the execution stack at that moment). The address for the binary code corresponding to the invocation `m1(53)` is denoted as ADDR1 and the address for the invocation `m1(x-20)` is ADDR2. We have shown the activation record for method `main`. Return address OS corresponds to the instruction of the operating system or of Eclipse that needs to be performed once the code finishes execution.

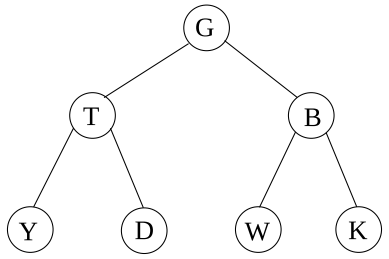
(A) (3 marks) What value will be printed for variable `r`? 82

(B) (5 marks) Draw the activation records in the execution stack:



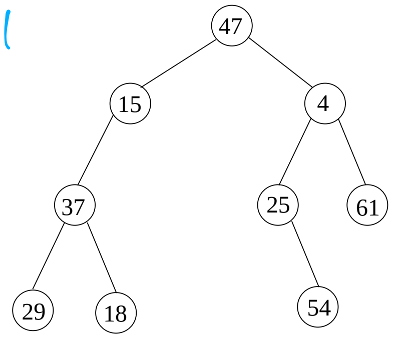
22. (5 marks) Consider the following code for a modified preorder traversal on a binary tree. Write the sequence of values printed when method `modPreorderIter` is invoked passing as parameter the root of the tree shown in the figure: G B K W T D Y

```
public void modPreorderIter (BinaryTreeNode<T> r) {
   if (r != null) {
       System.out.print(r.getData()+" ");
       modPreorderIter (r.getRight());
       modPreorderIter (r.getLeft());
   }
}
```



23. (5 marks) Answer the following questions for the tree shown below.

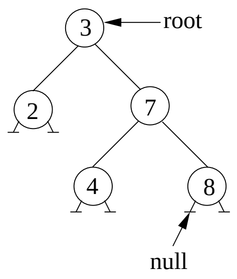
- Write the values in the nodes in the order in which they are visited by a preorder traversal: 47, 15, 37, 29, 18, 4, 25, 54, 61
- Write the values in the nodes in the order in which they are visited by an inorder traversal: 29, 37, 18, 15, 47, 25, 54, 4, 61
- Write the values in the nodes in the order in which they are visited by a postorder traversal: 29, 18, 37, 15, 54, 25, 61, 4, 47
- Write the values in the nodes in the order in which they are visited by a level order traversal: 47, 15, 4, 37, 25, 61, 29, 18, 54
- What is the degree of this tree? 2



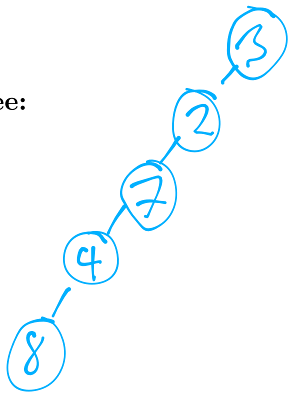
24. (7 marks) Consider the following algorithm

```
public void change(BinaryTreeNode node) {
   BinaryTreeNode left = node.leftChild(), right = node.rightChild();
   if (left != null && right != null) {
       change(left);
       change(right);
       left.setLeftChild(right);
       node.setRightChild(null);
   }
}
```

Draw the tree produced by the above algorithm when executed on the following tree, i.e. when the algorithm is invoked as `change(root)`.



Resulting tree:



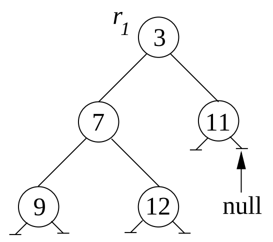
25. (14 marks) A *min heap* is a complete binary tree in which for every node other than the root, the value stored at that node is larger than or equal to the value stored in its parent. For example, the below tree with root r_1 is a min heap, but the tree with root r_2 is not a min heap because node p stores a value smaller than the value in its parent.

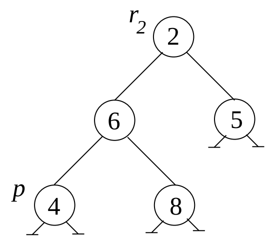
Write in Java or in detailed pseudocode like the one used in the lecture notes a **RECURSIVE** algorithm `isHeap`(`r`) that receives as parameters the root `r` of a complete binary tree and it returns *true* if the tree is a min heap and it returns false otherwise. Use `r.getValue()` to get the value stored in node `r`, `r.getParent()` to get the parent of `r` and `r.leftChild()` and `r.rightChild()` to get the children of node `r`.

<u>**Assume** that each internal node has 2 children.</u>

Algorithm isHeap(`r`)
In: Root `r` of a complete binary tree in which every internal node has 2 children
Out: True if the tree is a min heap; false otherwise

```
{
if (r == null)  return true;
if (r.getLeft()==null) return true;
int parentVal = r.getValue()
if (r.getLeft().getValue() < parentVal)
            return false;
if (r.getRight().getValue() < parentVal)
            return fase;
return  isHeap(r.getLeft()) &&
        isHeap(r.getRight());
}
```





26. (11 marks) Let `Q` be a queue storing a sorted sequence of integer values such that the smallest value is at the front of the queue. Write in Java or in detailed pseudocode like the one used in the lecture notes a **RECURSIVE** algorithm called reverse(Q) that reverses the order of the values stored in the queue, so at the end the values will be sorted in reverse order as before, i.e. the largest value will be stored at the front of the queue. The following figure shows an example of a sorted queue and the corresponding reversed queue.

Q

front	3	5	8	9	12	15	rear

reversed queue

front	15	12	9	8	5	3	rear

- The **ONLY** operations that you can use on the queue are `enqueue`, `dequeue`, and `isEmpty`.
- You **CANNOT** use any additional data structures (you cannot use another queue, stack, array, list and so on).
- You **CANNOT** assume that the queue is stored in an array, linked list or any other data structure; the only way to manipulate the queue is through the use of the operations `enqueue`, `dequeue`, and `isEmpty`.
- Your algorithm **CANNOT** have any loops (no for loops, while loops, do-while loops or any other loop statements).

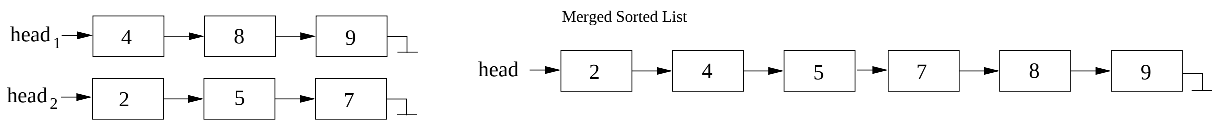
Hint. Remove the value at the front of the queue. Reverse the order of the remaining values in the queue. What do you do with the value that was removed from the queue?

Algorithm reverse(Q)
In: Queue Q storing a sorted sequence of integer values
Out: Nothing, but reverse the order of the values in the queue {

```
if (Q.isEmpty()) return true;
 int  v= Q.dequeue;
 reverse (Q);
  Q.enqueue(v);
}
```

27. (8 marks) **Optional question. Bonus marks.** Consider two sorted singly linked lists in which each node stores a DIFFERENT integer value and the values are sorted in increasing order. The first node of the first list is referenced by variable `head1` and it stores the smallest value in the first list. The first node of the second list is referenced by `head2` and it stores the smallest value in the second list.

Complete the following algorithm that merges the two lists into a single sorted list. Once the lists are merged variable `head` must point to the first node of the merged list, which has to store the smallest value in the list. Auxiliary variable `last` helps to know which is the last node of the merged list AS IT IS BEING CREATED. The following figure shows an example of two sorted list and the corresponding merged list.



Given a node `p` use `p.getNext()` and `p.setNext()` to get and set the next node in the list after `p`. Use `p.getValue()` to get the value stored in `p`.

Your answer must use **ONLY** the space provided, but you do not need to write code in each provided line, if you do not need to.

```
public Node merge(Node head1, head2) {
    Node head, last;
    if (head1.getValue() < head2.getValue()) {
        head = head1;
        head1 = head1.getNext();
    }
    else {
        head = head2;
        head2 = head2.getNext();
    }
    last = head;
    // Scan the lists
    while (head1 != null && head2 !=null) {                                    (a)

        if (head1.getValue() < head2.getValue()) { // Append node with smaller value to the list

            last.setNext(head1)                                                (b)

            head1 = head1.getNext();                                           (c)
        }

        else { last.setNext(head2);                                            (d)
            head2 = head2.getNext();                                           (e)
        }
        last = last.getNext(); // reference to the last node in the merged list   (f)
    }
    if (head1 != null)
        last.setNext(head1);                                                   (g)
    else last.setNext(head2);                                                  (h)
    return head;
}
```