Western University
Department of Computer Science

CS1027B Foundations of Computer Science II
Final Exam
April 24, 2023

Last Name: ____________________

First Name: ____________________

Student Number: ____________________

Section Number (1-Magguilli, 2-Sarlo, 3-Solis-Oba): __________

Instructions

- Fill in your name, student number, and section.
- The exam is 3 hours long and it has a total of 100 marks plus 8 bonus marks.
- The first part of the exam consist of multiple choice questions. For each question **CLEANLY AND DARKLY FILL IN ONLY ONE ANSWER**.
- For the second part of the exam, answer each question **ONLY IN THE SPACE PROVIDED**.
- Do not write in the top 1 inch of any page (where the barcode is shown).
- When you are done, raise your hand and one of the TA's will collect your exam.
- **YOU CANNOT LEAVE** the examination room in the last 15 minutes of the exam.

Part I. Multiple Choice Questions

For each multiple choice question clearly and darkly fill in the bubble for **ONLY ONE** answer.

1. (1 mark) Primitive local variables are stored in the execution stack and primitive instance variables are stored in the heap.

   [✓ True] ○ False

2. (1 mark) The smallest data item in a Binary Search Tree must be stored in the root node.

   ○ True [✓ False]

3. (2 marks) Consider the following class, `A`.

```
public class A {
    private int x;
    public A (int x) { this.x = x; }
}
```

   Determine what would happen from the following lines of code

```
A obj1 = new A(5);
A obj2 = new A(5);
Comparable<A> cmp = (Comparable<A>)obj1;
if (cmp.compareTo(obj2) == 0) System.out.println("Equal");
else System.out.println("Different");
```

   ○ There would be a compile-time error
   [✓ An exception would be thrown when the code is executed]
   ○ The code would compile and run without errors printing `Equal`.
   ○ The code would compile and run without errors printing `Different`.

4. (2 marks) Which of these statements is correct?

   ○ A class implementing an interface must provide code for **all** methods of the interface, including the `default` ones
   [✓ A class implementing an interface must provide code for all methods of the interface, except for the `default` methods for which an implementation is optional]
   ○ A class implementing an interface provides code for only the methods of the interface that the programmer selects
   ○ A class implementing an interface **must not** provide code for the `default methods`

5. (2 marks) Consider the following queue, `Q`, and code that corresponds to Queue `Q`.

   `Q` = 4 -1 7 5 -3 $\longleftarrow$ rear

```
while (!Q.isEmpty())
    if (Q.first() > 0) System.out.print(Q.dequeue() + " ");
```

   Determine which of the following statements is true.

   ○ This code would produce a compile-time error
   ○ This code would produce a run-time error
   [✓ This code would result in an infinite loop]
   ○ This code would result in the following output: **4 -1 7 5 -3**
   ○ This code would result in the following output: **4 7 5**

6. (1 mark) Consider this stack, `S` of type `ArrayStack<Integer>`, and code that corresponds to stack `S`.

$$\texttt{S} = \boxed{47 \quad 15 \quad 28 \quad 63} \longleftarrow \text{top}$$

```
ArrayStack<Integer> T = new ArrayStack<Integer>();
T = S;
S.pop();
System.out.print(T.peek());
```

Determine which of the following statements is true.

- This code would produce a compile-time error
- This code would produce a run-time error
- This code would result in the following output: **63**
- [✓ This code would result in the following output: **28**]

7. (3 marks) Consider the following code for a singly-linked list.

```
boolean done = false;
LinearNode curr = front;
int c = 0;
while (!done) {
    try {
         curr = curr.getNext();
         ++c;
    } catch (NullPointerException e) {
         done = false;
    }
}
System.out.println("Number of nodes = "+c);
```

Which of the following statements is true.

- This code would print the number of nodes in the list
- This code would print the number of nodes in the list minus 1
- The code would cause a compile-time error
- This code would crash during execution, so no value will be printed
- [✓ This code would be trapped in an infinite loop]

8. (2 marks) Consider a queue implemented using a circular array Q:

	0	1	2	3	4	5	6	7	8
Q				3	7	9	5		

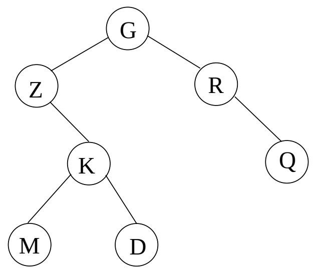
front = 3   rear = 6

Determine which of the following statements are valid ways to correctly update the `rear` variable in the enqueue() method of a queue implemented in this manner.

(i) `rear = rear + 1;`
(ii) `rear = (rear + 1) % Q.length;`
(iii) `rear = (rear - 1) % Q.length;`
(iv) `rear++; if (rear == Q.length) rear = 0;`
(v) `rear--; if (rear == -1) rear = Q.length - 1;`

- Only (ii)
- Only (iii)
- [✓ (ii) and (iv)]
- (iii) and (v)
- (i) and (ii)
- (i) and (iv)

Consider the following binary tree for the next three questions.



---

9. (1 mark) Determine the **preorder** traversal of the tree shown above.

○ Z, M, K, D, G, R, Q ○ G, Z, M, D, K, Q, R [✓ G, Z, K, M, D, R, Q]
○ G, Z, K, D, M, R, Q ○ G, Z, R, K, Q, M, D ○ M, D, K, Z, Q, R, G

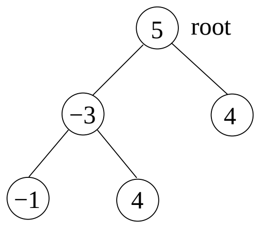
10. (1 mark) Determine the **inorder** traversal of the tree shown above.

[✓ Z, M, K, D, G, R, Q] ○ Z, K, M, D, G, R, Q ○ Z, M, K, D, R, Q, G
○ M, D, K, Z, G, R, Q ○ G, Z, R, K, Q, M, D ○ M, D, K, Z, Q, R, G

11. (1 mark) Determine the **postorder** traversal of the tree shown above.

○ Z, M, K, D, G, R, Q ○ G, Z, M, D, K, Q, R ○ G, Z, K, M, D, R, Q
○ M, D, K, Q, Z, R, G ○ M, D, K, Z, G, Q, R [✓ M, D, K, Z, Q, R, G]

12. (3 marks) Consider the following tree and the code that corresponds to the tree.



```
public int foo (BinaryTreeNode<Integer> node) {
    if (node == null) return 1;
    int p = 0;
    if (node.getDataItem() > 0) p = p + node.getDataItem();
    else p = p + 2;
    p = p + foo(node.leftChild());
    p = p + foo(node.rightChild));
    return p;
}
```

Determine what would be printed out if the following call to the above method was executed.

```
System.out.println(foo(root));
```

○ 8 ○ 9 ○ 15 ○ 20 [✓ 23] ○ 24

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 it returns `true` if every node of the tree stores a positive value and it returns false otherwise. Which of the following algorithms correctly solves this problem?

[✓ A1] ◯ A2 ◯ A3 ◯ A2 and A3 ◯ None of them ◯ All of them

```
public boolean Algorithm1(BinaryNode r) {
    if (r.getValue() <= 0) return false;
    else {
         if (r.leftChild() != null)
               if (Algorithm1(r.leftChild()) == false) return false;
         if (r.rightChild() != null)
               return Algorithm1(r.rightChild());
         return true;
    }
}

public boolean Algorithm2(BinaryNode r) {
    if (r.getValue() > 0) return true;
    else {
         if (r.leftChild() != null)
               if (Algorithm2(r.leftChild()) == true) return true;
         if (r.rightChild() != null) return Algorithm2(r.rightChild());
         return false;
    }
}

public boolean Algorithm3(BinaryNode r) {
    boolean tmp = true;
    if (r.getValue() <= 0) tmp = false;
    else {
         if (r.leftChild() != null) tmp = Algorithm3(r.leftChild());
         if (r.rightChild() != null) tmp = Algorithm3(r.rightChild());
    }
    return tmp;
}
```

14. (2 marks) Consider the following code fragment, where `iter` is an iterator storing the values 2, 4, and 5.

```
public void printStuff (Iterator<T> iter) {
    while (iter.next() != null) {
         System.out.print(iter.next() + " ");
    }
}
```

Which of the following statements are correct?

◯ The method printStuff() contains compile-time errors
[✓ The method printStuff() will compile but it will throw an exception]
◯ The method printStuff() will compile and run, but it will print only 2 values
◯ The method printStuff() will compile and print the three values

15. (2 marks) Consider the following method and determine which statement below is correct.

```
public void m() {
    ArrayQueue<String> q = new ArrayQueue<String>();
}
```

- ◯ `q` is stored in the static heap and the object referenced by `q` is stored in the dynamic heap
- [✓ `q` is stored in the execution stack and the object referenced by `q` is stored in the dynamic heap]
- ◯ `q` is stored in the dynamic heap and the object referenced by `q` is stored in the execution stack
- ◯ `q` and the object referenced by `q` are both stored in the dynamic heap
- ◯ `q` and the object referenced by `q` are both stored in the execution stack

16. (3 marks) Which of the following arrays represents a min heap?

A1

3	5	7	4	6	9

A2

5	6	8	7	9	9

A3

4	7	6	8	9	5

◯ A1 [✓ A2] ◯ A3 ◯ A1 and A2 ◯ A1 and A3 ◯ A2 and A3
◯ None ◯ All of them

17. (2 marks) The following array represents a complete binary tree. Does it represent a binary search tree?

[✓ Yes] ◯ No

7	5	9	3	6	8	11	2

18. (4 marks) Consider the following code.

```
public void sort(int[] a, int n) {
    int j = 0;
    while (j < n - 1) {
        int index = j;
        (**)
        int y = a[index];
        a[index] = a[j];
        a[j] = y;
        ++j;
    }
}
```

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

- ◯ `for (int i = j+1; i < n; i = i + 1) if (a[i] < a[index]) index = a[i];`
- ◯ `for (int i = j+1; i < n; i = i + 1) if (a[i] > a[index]) index = a[index];`
- ◯ `for (int i = j+1; i < n; i = i + 1) if (a[i] < a[index]) index = i;`
- [✓ `for (int i = j+1; i < n; i = i + 1) if (a[i] > a[index]) index = i;`]
- ◯ `for (int i = 0; i < n; i = i + 1) if (a[i] < a[index]) index = i;`
- ◯ `for (int i = 0; i < n; i = i + 1) if (a[i] > a[index]) index = i;`

Questions 19, 20, and 21 are OPTIONAL and they represent bonus marks.

19. (3 marks) Consider the following Java program.

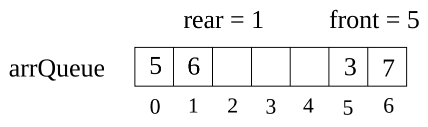
```
public static void m(int size) {
    if (size == 0) return;
    else m1(size-1);
    m1(size);
}
```

```
public static void m1(int s) {
    int i = s-2;
    if (s <= 1) return;
    else m(i);
}
public static void main(String[] args) {
    m(2);
}
```

An activation record for method `m1` uses 10 bytes, an activation record for method `m` uses 10 bytes, and an activation record for `main` uses 10 bytes. How much memory is needed for the execution stack when the program is executed? The amount of memory needed is equal to the total size of the maximum number of activation records that are in the execution stack at the same time.

○ 10 bytes ○ 20 bytes ○ 30 bytes [✓ 40 bytes] ○ 50 bytes

20. (2 marks) Consider an empty stack `s` and a queue `q` storing the following values: 3, 7, 5, and 6. The queue is implemented using a circular array `arrQueue` where public instance 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

```
for (int i = 0; i < 4; i = i+1) s.push(i);
for (int i = 1; i <= 3; i = i + 1) {
    q.enqueue(s.pop());
    if (((q.rear + 1) % 7) != q.front) q.enqueue(s.pop());
    s.push(q.dequeue());
}
```

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

○ `front = 0, rear = 6` [✓ `front = 1, rear = 6`] ○ `front = 1, rear = 5`
○ `front = 6 rear = 5` ○ None. The code will throw an exception

21. (3 marks) The following algorithms compute the `n`-th Fibonacci number, for `n > 2`.

```
public int ifib(int n) {
    int f1 = 1, f2 = 1, f3 = 2;
    for(int i = 3; i <= n; ++i) {
         f3 = f1 + f2; f1 = f2; f2 = f3;
    }
    return f3;
}
```

```
public int rfib(int n) {
    if (n == 1 || n == 2) return 1;
    else return rfib(n-1) + rfib(n-2);
}
```

Which of the following statement is true assuming that `n` is very large?

○ Both algorithms solve the same problem, so both take the same amount of time to execute
○ Algorithm `ifib` is slower because it must waste time storing values in variables `f1`, `f2`, and `f3`
○ Algorithm `rfib` is faster because recursion is very fast
[✓ Algorithm `rfib` is a lot slower because of the large number of recursive calls that it makes.]

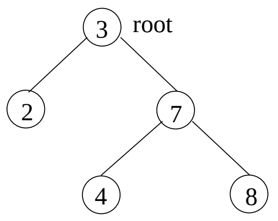
Part II. Written Answers

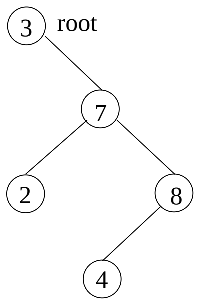
22. (6 marks) Consider the following algorithm.

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





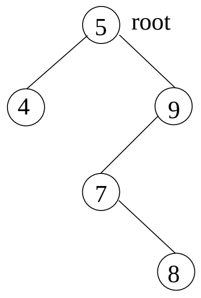
23. (5 marks) Consider the following min heap. Delete the minimum value from the min heap and show the resulting min heap in the empty array provided. You must use the algorithm studied in class for removing the minimum value from a heap.

Hint. If you do not remember the algorithm, it first removes the minimum value replacing it with the last value in the min heap. Then, repeatedly it traverses down the heap comparing the value stored in a node with the smaller value in its children, swapping values if needed.

Min heap	2	3	15	6	20	29	22	8	17
	0	1	2	3	4	5	6	7	8

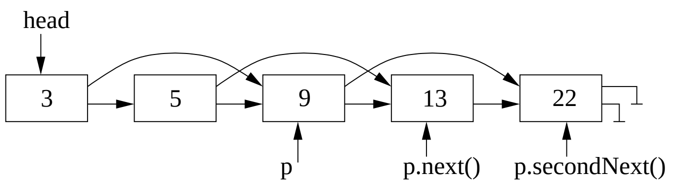
Resulting min heap	3	6	15	8	20	29	22	17	
	0	1	2	3	4	5	6	7	8

24. (4 marks) Draw a **binary search tree** storing the values 4, 5, 7, 8, 9 such that a preorder traversal of the tree visits the nodes in the order: 5, 4, 9, 7, 8.



For the following two questions write algorithms in Java or in detailed Java-like pseudocode like the one used in the lectures. **Use ONLY the space provided to write your answers.**

25. A *twofold* list is a linked list in which every node has two pointers: One to the next node in the list and one to the next node of the next node in the list. The following figure shows an example of a twofold list. Given a node `p` in a twofold list, `p.next()` is the next node in the list after `p` and `p.secondNext()` is the next node of the next node after `p`. Each node stores a different integer value, and the values are stored in the nodes of the list in increasing order.



- (4 marks) Complete algorithm `findSecondPrev(head, target)` that receives as parameters: `head` pointing to the first node of a twofold list, and an integer value `target` and it must return the node `p` such that `p.secondNext().getData() > target` and `p.getNext().getData() < target`.

  Note. For simplicity assume that `target` is larger than the values stored in the first two nodes of the list and it is smaller than the value stored in the last node, so above node `p` always exists.

```
Algorithm findSecondPrev(head, target)
In: First node of a twofold list, and integer target larger than the values stored in the first two
    nodes and smaller than the value in the last node.
Out: Node p as described above.
p = head
while not((p.getNext().getData() < target) and (p.secondNext().getData() > target))
do {
    p = p.getNext()
}
return p
```

- (5 marks) Complete the following algorithm `add(head, target)` that adds a node storing the value `target` to the list, so at the end the values are stored in the nodes of the list in increasing order. Use `setNext` and `setSecondNext` to change the pointers of a node.

  Assume again that `target` is larger than the values stored in the first two nodes of the list and smaller than the value stored in the last node.

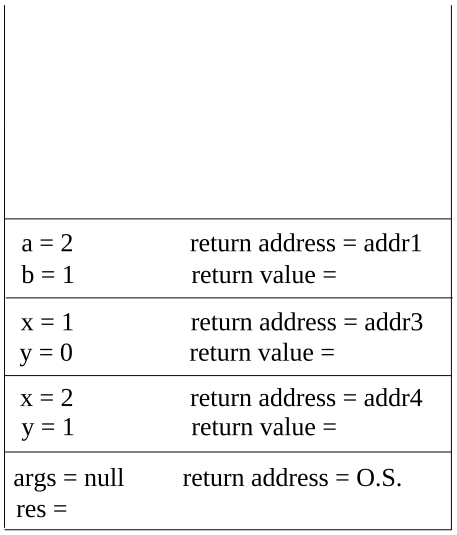
```
Algorithm add(head, target)
In: First node of a twofold list, and integer target.
Out: Add target to the list, so at the end the values are stored in the nodes in increasing order.
newNode = new node storing target
p = findSecondPrev(head,target)
n = p.getNext()
s = p.secondNext()

newNode.setNext( s  or n.getNext()  or p.secondNext() )

newNode. setSecondNext(s.getNext())  or setSecondNext(n.secondNext())

p. setSecondNext(newNode)

n. setNext(newNode)  or setNext(p.secondNext())

n. setSecondNext(s)  or setSecondNext(newNode.getNext())
```

Hint. `newNode` must be added between which two nodes?

26. Consider the following code fragments. Hand-trace through the code and determine what the execution stack would contain **JUST BEFORE** the line that is identified with asterisks and says `"PAUSE EXECUTION HERE"` is executed. At that point in the execution of the program, draw what the execution stack would contain including any relevant information in the activation records. Then continue hand-tracing to determine the final result that would be printed out. Only one execution stack has to be drawn at the instant in which the above line is reached. The activation record for method `main` has been drawn for you. Use the addresses `addr1` – `addr4` as labelled in the code for methods invocations' addresses.

```
public static int run(int a, int b) {
    if (a == b) return 1;
    else {
         a = b + 4;          // *** PAUSE EXECUTION HERE ***
         return a * 2;
    }
}
public static int proc(int x) {
    int y = 0;
    if (x == 1) return run(2,1); addr1
    else if (x == 2) y = run(1,1); addr2
    if (y == 1) return proc(y); addr3
    return y;
}
public static void main(String[] args) {
    int res = proc(2); addr4
    System.out.println(res);
}
```

(a) (9 marks) Fill in the execution stack at the instant just before the line with the comment `"PAUSE EXECUTION HERE"` is executed.



a = 2
b = 1
	return address = addr1
	return value =

x = 1
y = 0
	return address = addr3
	return value =

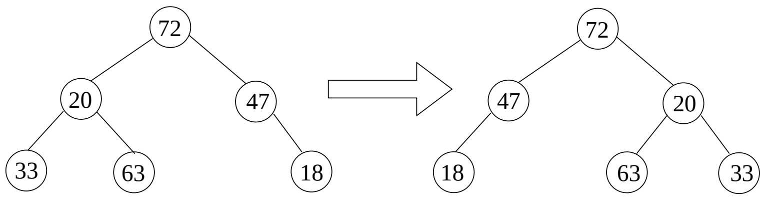
x = 2
y = 1
	return address = addr4
	return value =

args = null res =	return address = O.S.

(b) (2 marks) Determine the final value, `res`, that is printed out: 10

27. (14 marks) Write in Java or in **detailed pseudocode** like the one used in the lecture an algorithm `flip(r)` that receives as parameter `r` which is a reference to a node (initially to the root node) of a binary tree and that "horizontally" flips the tree so that it becomes a mirror image of its original state, as shown in the example below. The algorithm must work for **ANY** binary tree, not just the example shown below. The algorithm **MUST** be recursive. You can use the methods `getLeft()`, `getRight()`, `setLeft()`, and `setRight()` on any non-null node in the tree.

Hint: Swap the children of every internal node.



```
**Algorithm** flip(r)
**In:** Node r of a binary tree
**Out:** Nothing, but flip the tree "horizontally" so that it is now the mirror image of its original state
if r ≠ null then {
   flip(r.getLeft()
   flip(r.getRight())
   tmp = r.getLeft()
   r.setLeft(r.getRight())
   r.setRight(tmp)
}
```

Here is another solution:

```
Algorithm flip(r)
if r ≠ null then {
   tmp = r.getLeft()
   r.setLeft(r.getRight())
   r.setRight(tmp)
   flip(r.getLeft()
   flip(r.getRight())
}
```

28. (14 marks) Write in Java or in **detailed pseudocode** like the one used in the lecture notes a **RECURSIVE** algorithm `find(target,S)` that receives as parameter an integer value `target` and a stack `S` storing integer values such that the algorithm returns `true` if `target` is stored in the stack, and it returns `false` otherwise; furthermore, at the end of the execution of the algorithm **THE STACK MUST BE IDENTICAL** as to how it was before the algorithm was executed, this means that at the end `S` must contain the same values and in the same order as they were in it before the algorithm was executed.

Your algorithm **MUST** satisfy the following conditions:

- Your algorithm **CANNOT HAVE ANY LOOPS**.
- The **ONLY methods** that you can use in your algorithm to manipulate the stack are `push()`, `pop()`, `peek()`, and `isEmpty()`.
- You **CANNOT** use any auxiliary data structures (you cannot use an array, a second stack, a queue, a linked list, an ArrayList, and so on).
- You **CANNOT** assume that the stack is implemented using an array, singly linked list, doubly linked list, or any other data structure. Hence, you **CANNOT**, for example, write something like `S[i] = ⋯` or `something = S[j]`.

If you do not follow the above conditions, you will receive ZERO marks for your answer.

Hint. Take a value from `S` and check if it is the required one. If not, search for `target` in the rest of the stack. What do you do with the value that you took from the stack to ensure that the stack is exactly the same when the algorithm finishes?

```
Algorithm find(target,S)
In: Integer value target and stack S storing integer values
Out: true if target is in S; false otherwise. The stack S must be identical to how it was before
the algorithm was executed.
if S.isEmpty() then return false
else {
   if S.peek() = target then return true
   else {
      val = S.pop()
      res = find(target,S)
      S.push(val)
      return res
}
```

Another solution:

```
Algorithm find(target,S)
if S.isEmpty() then return false
else {
   val = S.pop()
   if val = target then {
      S.push(val)
      return true
   }
   else {
      res = find(target,S)
      S.push(val)
      return res
   }
}
```