This is a closed-book, closed-neighbor, closed-internet, open-mind test. You are allowed one page of notes. The test has 6 pages; make sure you have them all. Write your name on the exam. There are 8 questions worth a total of 100 points. Answer all questions in the space provided below, or alongside, each question. Make it clear what your answer is. The space provided should prove sufficient, but if you happen to run out of space, continue on the back of the page. If you need scratch space, use the backs of pages.

(1 - 4): Multiple choice (20 pts, 5 ea.)

Note that all questions ask you to select all answers that apply.

1. Suppose \( \text{arr} \) is an int array, initialized with \( \text{int[]} \ arr = \{11, 22, 33, 44, 55, 66, 77\} \). Which of the following are true about using binary search on \( \text{arr} \)? Select all that apply:
   (a) it will require at least 6 comparisons to find out that 68 is not in \( \text{arr} \),
   (b) a search for 66 will find it in 2 comparisons,
   (c) a search for any element will begin with a comparison with 44,
   (d) a search for 43 will require fewer comparisons than a search for 2017,
   (e) binary search won’t work on \( \text{arr} \) because it’s already sorted.

2. Suppose \( \text{arr} \) is an int array, initialized with \( \text{int[]} \ arr = \{1, 2, 3, 4, 5, 6, 7, 8\} \). Which of the following are true about using mergesort on \( \text{arr} \)? Select all that apply:
   (a) the recursion will go 5 or more levels deep,
   (b) fewer recursive calls will be made than if the array was not in order,
   (c) the final merge step will start with comparing 1 and 5,
   (d) the first merge step will begin with comparing 1 and 8
   (e) mergesort won’t work on \( \text{arr} \) because it’s already sorted.

3. Which of the following are true about a binary search tree (BST)? Select all that apply: (a) all the keys to the left of the root compare less than the key at the root, (b) all the keys to the left of the root compare less than any key on the right of the root, (c) the height of the tree depends on the order in which the keys were inserted, (d) the height of the tree depends on the \textit{values} inserted for each key, (e) if a tree of \( N = 2^x - 1 \) elements is perfectly balanced, the middle key (the one with \( 2^x - 1 \) keys less than it and \( 2^x - 1 \) keys more than it) can be found in constant time.

4. Which of the following statements are true? Select all that apply. (a) If an array is suitable for binary search, the largest element in the array can be found in constant time, (b) In a linked list, it’s OK to have more than one \texttt{next} reference be null, (c) If a linked list is in sorted order, you can use binary search on it, (d) Mergesort on an array of 1024 elements does more than 1000 assignments, but fewer than 100,000.

(e) Finding the smallest key in a binary search tree of \( N \) elements will have quadratic \( (N^2) \) order of growth.
5. (15pts) The function `whatdoIdo()` below takes a reference to a linked list and returns an integer.

```java
class Node {
    String data;
    Node next;
}

int whatdoIdo(Node first) {
    if (first == null) {
        return 0;
    }
    return 1 + whatdoIdo(first.next);
}
```

(a) Describe how this function works and what the return value represents. You are encouraged to draw pictures to illustrate your description.

(b) What order of growth would you expect to see for this function? Explain your answer briefly.
6. (20pts) The function `whatdoIdo()` below takes a reference to a linked list and modifies that list.

```java
class Node {
    String data;
    Node next;
}

void whatdoIdo(Node first) {
    Node current = first;
    while(current != null) {
        Node temp = new Node();
        temp.data = current.data;
        temp.next = current.next;
        current.next = temp;
        current = temp.next;
    }
}
```

(a) Describe how this function works and what the list looks like after the function is called. You are encouraged to draw pictures to illustrate your description.

(b) What order of growth would you expect to see for this function? Explain your answer briefly.

(c) What would happen if the line `current = temp.next;` were changed to: `current = current.next;`
7. (20 pts) This problem concerns creating a function

```
int lastPos(String s, Node first)
```

which takes a String and a linked list of Node elements (defined in question 5), and
returns the position of the last item in the list which matches the string, or -1 if the list is
empty or the string is not in the list at all. See the illustrations below for examples.

Given a list with 8 elements: “this” “is” “a” “very” “very” “very” “long” “list”,
```
lastPos(“this”, first) returns 0
lastPos(“a”, first) returns 2
lastPos(“very”, first) returns 5
lastPos(“list”, first) returns 7
lastPos(“not”, first) returns -1
```

(a) Briefly describe in plain English your plan for how to do this.

(b) Write the function as Java code, including comments as you see fit.

(c) What order of growth would you expect to see for this function? Explain your answer
briefly.
8. (25 pts) Given below is a method for binary search tree traversal that uses a queue of BST nodes for its operation. The following page shows the code for this method.

1) Put the root node of the tree into the queue.
2) While the queue is not empty,
   2a) Dequeue a node from the queue,
   2b) if the node has a non-null left child, enqueue the node’s left child,
   2c) if the node has a non-null right child, enqueue the node’s right child,
   2d) print out the key of the node removed in step 2a.

(a) For the tree below, in what order will the keys be printed by this method?

(b) What would the order be if the queue were replaced with a stack?
public class MyBST<Key extends Comparable<Key>, Value> {
    private Node root; // root of BST

    private class Node {
        private Key key; // sorted by key
        private Value val; // associated data
        private Node left, right; // left and right subtrees
    }

    public void printUsingQueue() {
        Queue<Node> q = new Queue();
        q.enqueue(root);
        while(! q.isEmpty()) {
            Node n = q.dequeue();
            if (n.left != null) {
                q.enqueue(n.left);
            }
            if (n.right != null) {
                q.enqueue(n.right);
            }
            StdOut.print(n.key + " ");
        }
        StdOut.println();
    }

    public void printUsingStack() {
        Stack<Node> s = new Stack();
        s.push(root);
        while(! s.isEmpty()) {
            Node n = s.pop();
            if (n.left != null) {
                s.push(n.left);
            }
            if (n.right != null) {
                s.push(n.right);
            }
            StdOut.print(n.key + " ");
        }
        StdOut.println();
    }
}