FINAL EXAMINATION
DECEMBER 2015
CSC 212 ♦ SECTION 01
INSTRUCTOR: NICHOLAS R. HOWE

YOU MAY USE TWO 8.5"x11" SHEETS OF NOTES ON THIS EXAM.
YOU MAY NOT USE THE TEXTBOOK, A COMPUTER, OR ANY OTHER INFORMATION SOURCE BESIDES YOUR TWO PAGES OF NOTES.

All work should be written in the exam booklet. Partial credit will be granted where appropriate if intermediate steps are shown.

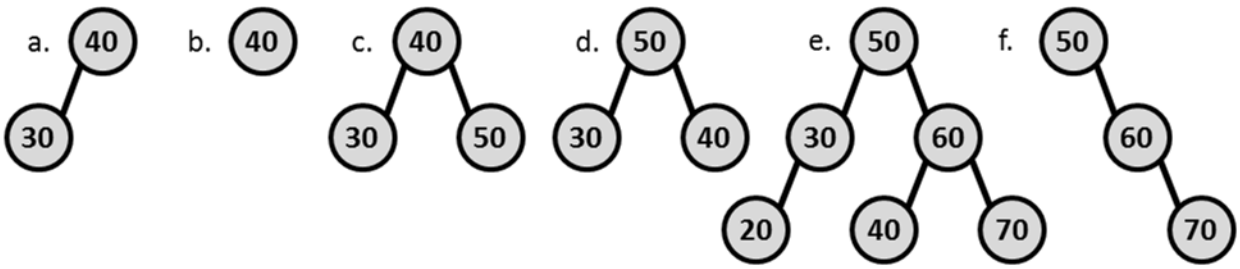
Data Structure Operations (16 points)

For each of the operations below, give the best possible big-O bound on the running time.

a.) Look up the i^{th} element of an array.
b.) Look up the i^{th} element of a linked list.
c.) Insert an element at the front of an array
d.) Insert an element at the front of an linked list
e.) Insert an element into a heap
f.) Insert an element into a hash table
g.) Insert an element into a balanced binary search tree
h.) Sort an array of integers
i.) Sort a linked list of integers
j.) Find the index of item q in a sorted array
k.) Find the index of item q in a sorted linked list
l.) Determine whether the value q is stored within a balanced binary search tree
m.) Determine whether a given key is stored in a hash table
n.) Insert item q immediately before the i^{th} element of an array
o.) Insert item q immediately before the i^{th} element of a linked list
p.) Insert item q immediately before the i^{th} element of a binary search tree

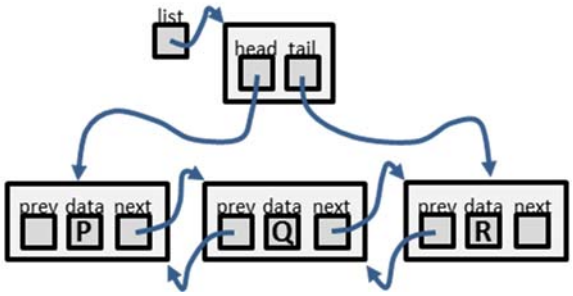
Trees (12 points)

Determine whether each of the trees shown below meets the definition of (i) a binary search tree, (ii) a heap, (iii) both, or (iv) neither.

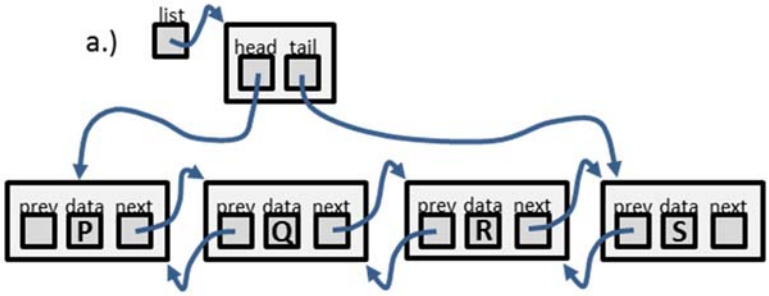


Lists, Stacks & Queues (16 points)

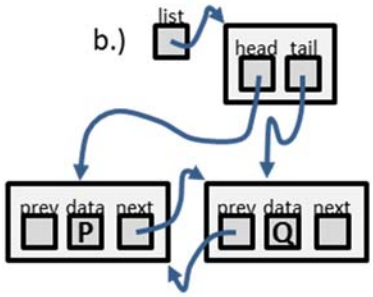
Consider the list shown below. For the figures that follow, identify which common stack and queue operation(s), if any, could have led to the situation shown.



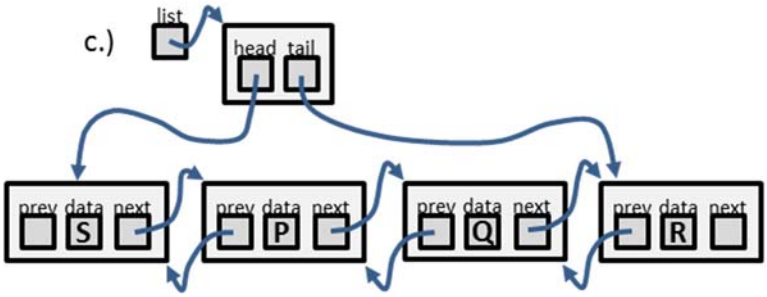
a.)



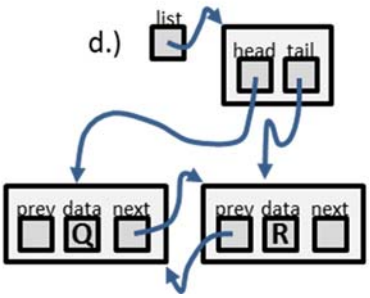
b.)



c.)

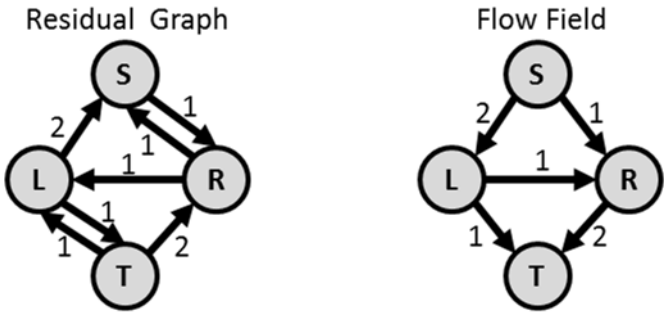


d.)



Graphs (12 points)

The figure below shows the condition of a network during a computation of the maximum flow. On the left is a residual graph, and on the right is the corresponding flow field.



a.) Draw the original network under zero flow, showing all the initial capacities.

b.) Identify an augmenting path in the residual graph shown.

c.) Show the flow field resulting from adding your augmenting path to the existing flow shown.

Hash Tables (8 points)

We mentioned in class that hash table sizes should generally be prime numbers. This question asks you to imagine what could happen if one ignores this advice.

Suppose that we create a hash table with exactly 1000 rows. We want to store objects using their hashCode as the key. Recall that many implementations of Java use the memory address as the hashCode. For reasons not important to this question, many operating systems assign variables to memory addresses that are multiples of the system *word size*. Given a word size of 8, the hashCode of an arbitrary object will always be a multiple of 8.

a.) How many lines of the table will never be used if all hash codes are multiples of 8?

b.) If the word size changed to 16, how many lines could never be used?

c.) If the table size is changed to 997 (a prime number), how many lines could never be used?

d.) The String class redefines the hashcode so that it is based on the characters in the string, not the memory address. Would using a table of 1000 rows possibly cause any problems in this case?

Recursion (10 points)

Consider the recursive method shown below, and answer the questions that follow.

```
public static int bsearchR(int q, int[] arr) {                  // line 1
  return bsearchR(q,arr,0,arr.length);                          // line 2
}                                                               // line 3
public static int bsearchR(int q, int[] arr, int lo, int hi) {  // line 4
  System.out.println("R:"+lo+" "+hi);                          // line 5
  if (lo>=hi) {                                                 // line 6
      return lo;                                                // line 7
  } else {                                                      // line 8
      int p = (lo+hi)/2;                                        // line 9
      if (q <= arr[p]) {                                       // line 10
        return bsearchR(q,arr,lo,p);                           // line 11
      } else {                                                 // line 12
        return bsearchR(q,arr,p+1,hi);                         // line 13
      }                                                        // line 14
  }                                                            // line 15
}                                                              // line 16
```

a.) Which line(s) contain the stop condition?

b.) Which line(s) contain the recursive step?

c.) In what way is does the recursive call simplify the problem?

d.) Characterize the running time of this method, using big-O notation, in terms of the array length **n**.

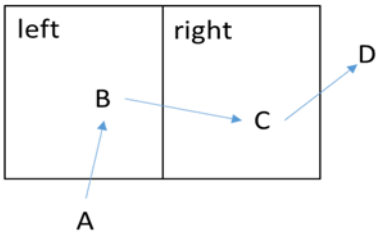
d.) Rewrite the same computation as an iterative algorithm.

Graphical User Interfaces (10 points)

Suppose that a GUI interface window consists of two side-by-side **JButton** elements (called **left** and **right**) as shown in the illustration. Several listeners have been registered, as shown below:

```
left.addMouseListener(new MouseListener1());
right.addMouseMotionListener(new MouseMotionListener2());
right.addActionListener(new ActionListener3());
```

Suppose further that the mouse begins at the point labeled A. The user moves it to point B and clicks once, then moves it to point C, clicks again, and finally moves it to point D. List the sequence of event handler methods that will potentially be called, in the order they will be called. If a handler might be called more than once in a row, indicate the number of times if it is known.



Java Language (16 points)

Consider the program shown below. Draw a diagram showing the state of memory at the point where the statement labeled NOW has just been executed. Your diagram should follow the style of the ones drawn in class. It should show all local variables, with the regions corresponding to the call stack and the heap clearly denoted. The value currently stored in each variable should be shown. Any memory that was previously allocated but is now eligible for garbage collection should be shown in a separate section of the heap, labeled GC. You need not show any static variables, and you may ignore the args parameter to main.

```
public class Diagram2 {
    private int a;
    private Diagram2 d;
    public Diagram2(int a, Diagram2 d) {
         this.a = a;
         this.d = d;
    }
    public static void extend(Diagram2 d) {
         d.d = new Diagram2(d.a+1,d.d);
    }
    public static void test(Diagram2 d) {
         if (d.d != null) {
             test(d.d);
         }
         System.out.println(d.a);  // NOW
    }
    public static void main(String[] args) {
         Diagram2 d = new Diagram2(5,new Diagram2(8,null));
         extend(d.d);
         extend(d);
         test(d.d);
    }
}
```