Comp 250: Final examination - VERSION #1

Instructors: Mathieu Blanchette and Jérôme Waldispühl

December 21st, 2015, 2:00 - 5:00pm

Examiner's signature:

Associate examiner's signature:

NAME:

STUDENT ID:

NOTE: DO NOT ANSWER ON THIS COPY.

- This is a multiple choices exam. For each question, only one answer can be provided.
- Answer the questions on the multiple choice page, using a lead pencil.
- You have 180 minutes to write the exam.
- This exam is worth 50% of your total mark.
- You can use up to 2 pages (1 double-sided sheet) of notes.
- Books and electronic devices are not allowed.
- If you believe that none of choices provided for a given question are correct, provide the answer that is the closest to being correct.
- This exam contains 45 questions on 20 pages.
- This examination is printed on both sides of the paper
- This examination paper must be returned
- The examination Security Monitor Program detects pairs of students with unusually similar answer patterns on multiple-choice exams. Data generated by this program can be used as admissible evidence, either to initiate or corroborate an investigation or a charge of cheating under Section 16 of the Code of Student Conduct and Disciplinary Procedures.

1. (4 points) Consider a sorted array of n distinct integers. What will be the complexity of the best algorithm to find the number of pairs of elements in the array whose sum is equal to 100?

   A. $\Theta(n)$

   B. $\Theta(n\log(n))$

   C. $\Theta(n^2)$

   D. $\Theta(n^2\log(n))$

   E. $\Theta(100)$

2. (4 points) What will be the complexity of the following $count(n)$ algorithm:

Algorithm 1 Count(int n)

```
int count=0;
for int i = n; i > 0; i=i/2 do
    for int j = 0; j < i; j++ do
        count++;
    end for
end for
return count
```

   A. $\Theta(\log(n))$

   B. $\Theta(n)$

   C. $\Theta(n\log(n))$

   D. $\Theta((n\log(n))^2)$

   E. $\Theta(n^2)$

Trees in Java

In assignment 4, you wrote a program to calculate the derivative of an algebraic expression represented by an expression tree. The operations supported included "add", "mult", "minus", etc. Suppose we want to allow one more type of operations, called "square", where $square(f(x)) = (f(x))^2$. Recall that $\frac{d(f(x)^2)}{dx} = 2 \cdot f(x) \cdot \frac{df(x)}{dx}$. Which lines of code complete the portion of code that would need to be added to the differentiate method you implemented in order to handle this new type of operation?

```
class expressionTreeNode {
    private String value;
    private expressionTreeNode leftChild, rightChild, parent;

// constructor
  expressionTreeNode(String s, expressionTreeNode l,
                        expressionTreeNode r, expressionTreeNode p) {
        value = s;  leftChild = l;  rightChild = r;  parent = p;
    }

   expressionTreeNode getLeftChild() { return leftChild; }
   expressionTreeNode getRightChild() { return rightChild; }

expressionTreeNode differentiate() {
/* Same beginning as in your assignment solution */
...
       if ((getValue()).equals("square")) {
            expressionTreeNode l,r;
            l=new expressionTreeNode("2",null,null, null);
            // LINE #1
            // LINE #2
        }
}
```

3. (4 points) Which line correctly completes LINE #1?

   A. r= new expressionTreeNode("mult", getLeftChild().deepCopy(), getLeftChild().differentiate(),null);

   B. r= new expressionTreeNode("mult", getLeftChild().differentiate(), getLeftChild().differentiate(),null);

   C. r = new expressionTreeNode("mult", getLeftChild().deepCopy(), getLeftChild().deepcopy(),null);

   D. r = new expressionTreeNode("mult", getLeftChild().differentiate(), getRightChild().deepCopy(),null);

   E. r = new expressionTreeNode("mult", getLeftChild().differentiate(), getRightChild().differentiate(),null);

4. (4 points) Which line correctly completes LINE #2?

   A. return new expressionTreeNode("add", l, r, null);

   B. expressionTreeNode("square", l, r, null);

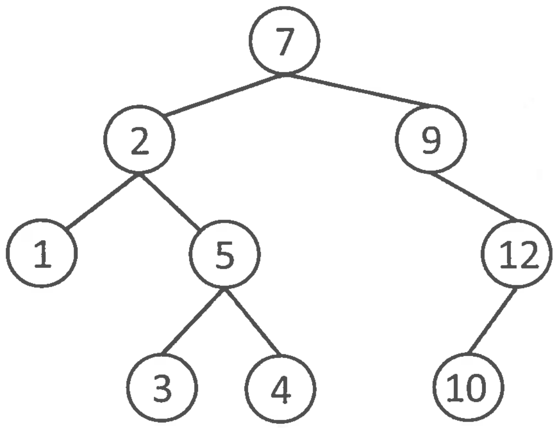
   C. return expressionTreeNode("add", l, r, null);

D. return expressionTreeNode("mult", l, r, null);

E. None of the above

Trees traversals

5. (4 points) Consider the following binary search tree:



What is the post order traversal for the given binary search tree ?

A. 7, 2, 9, 1, 5, 12, 3, 4, 10

B. 7, 2, 1, 5, 3, 4, 9, 12, 10

C. 1, 2, 3, 5, 4, 7, 9, 10, 12

D. 1, 3, 4, 5, 2, 10, 12, 9, 7

E. None of the above

6. (4 points) Consider the following pair of recursive algorithms calling each other to traverse a binary tree.

Algorithm 2 WeirdPreOrder(treeNode n)

```
if (n ≠ null) then
    print n.getValue()
    weirdPostOrder(n.getRightChild())
    weirdPostOrder(n.getLeftChild())
end if
```

Algorithm 3 WeirdPostOrder(treeNode n)

```
if (n ≠ null) then
    weirdPreOrder(n.getRightChild())
    weirdPreOrder(n.getLeftChild())
    print n.getValue()
end if
```

Which one of the following is the output being printed when weirdPreOrder(root) is executed on the binary tree shown above?

A. 7 12 10 9 5 4 3 1 2

B. 7 2 1 4 3 4 9 12 10

C. 1 2 3 5 4 7 9 10 12

D. 7 9 10 12 5 4 3 2 1

E. None of the above

Stacks

Consider the recursive algorithm below.

Algorithm 4 mistery(Stack stack)

```
value ← stack.pop()
if stack.isempty() then
    return value
else
    result ← mistery(stack)
    stack.push(value)
    return result
end if
```

We want to discover the purpose of this algorithm. First, we execute the following commands.

```
stack = New Stack();
stack.push("1");
stack.push("2");
stack.push("3");
print mistery(stack);
print mistery(stack);
print mistery(stack);
```

7. (4 points) What are the values printed during the execution of the instructions above.

   A. 3 2 1

   B. 2 1 3

   C. 1 2 1

   D. 1 2 3

   E. 3 3 3

8. (4 points) What is this algorithm doing?

   A. It build an array-based representation of a heap with the numbers inserted in the stack.

   B. It simulates the removeMin() operation on a min heap.

   C. It removes and returns the element at the bottom of the stack.

   D. It sorts the numbers inserted in the stack.

E. It changes the content of the stack by reversing the order of its elements.

9. (4 points) If the stack contains n elements, what is the worst-case running time complexity of mistery()?

   A. $\Theta(1)$

   B. $\Theta(\log n)$

   C. $\boldsymbol{\Theta(n)}$

   D. $\Theta(n \cdot \log n)$

   E. $\Theta(n^2)$

Binary Search Trees

We propose the following algorithm to insert new values in a binary search tree. Keys are integers.

Algorithm 5 rootinsert(Node root, int newvalue)

```
newnode = new Node()
newnode.setValue(newvalue)
if newvalue ≤ root.getValue() then
    newnode.setRight(root)
    newnode.setLeft(root.left)
    root.setLeft(NULL)
else
    newnode.setLeft(root)
    newnode.setRight(root.right)
    root.setRight(NULL)
end if
return newnode
```

10. (4 points) This algorithm is supposed to return a binary search tree in which we added a new key. Which of the following statements is correct?

    A. This algorithm is correct and runs in $\mathcal{O}(1)$.

    B. This algorithm is correct but produces trees with internal nodes having only one child.

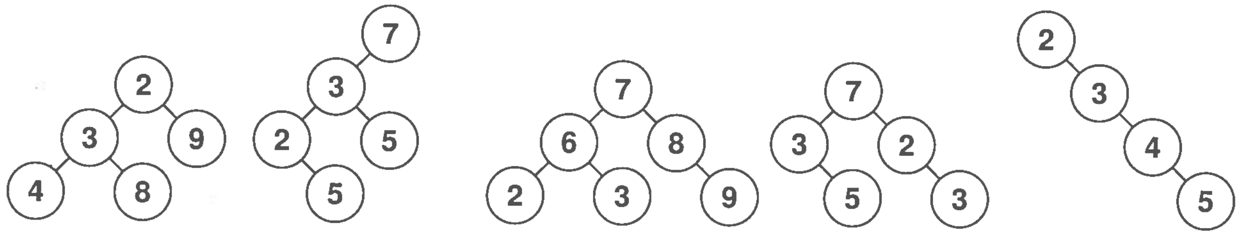
    C. This algorithm works if and only if the key that is being inserted has a value equal to the min or max value stored in the input binary search tree.

    D. This algorithm returns a valid binary search tree if there does not exist a key k such that $root.getValue() \leq k \leq newvalue$ **or** $newvalue \leq k \leq root.getValue()$.

    E. None of the the above.

Binary trees

Consider the following trees.



(A) (B) (C) (D) (E)

11. (4 points) Which of the tree above is a binary search tree? A. B. C. D. E.

12. (4 points) Which of the tree above is a heap? A. B. C. D. E.

Recursive algorithms

Consider the following recursive algorithm:

Algorithm 6 TestRecursive(int n)

Require: A non-negative integer n.

```
if (n = 0) then
    print "Zero"
    return
else
    TestRecursive( ⌊n/2⌋)
end if
return
```

13. (4 points) What will be printed when testRecursive(12) is executed? Assume that everything that gets printed appears on the same line.
    A. Zero 1 3 6 12
    B. 12 6 3 1 0 Zero
    C. 12 6 3 1 Zero
    D. Zero 0 1 3 6 12
    E. None of the above

14. (4 points) Suppose we are using the ProgramStack class seen in assignment 3 to use a stack to model recursion. What is the content of the ProgramStack at the point of the execution where "Zero" gets printed, during the execution of testRecursive(2)? Assume that the top of the stack is shown on the left.

A. [PC= 2, n=0], [PC= 5, n=1], [PC= 5, n=2]

B. [PC= 5, n=2], [PC= 5, n=1], [PC= 2, n=0]

C. [PC= 2, n=2], [PC= 1, n=1], [PC= 0, n=0]

D. [PC= 2, n=2], [PC= 5, n=2], [PC= 5, n=2]

E. None of the above

15. (4 points) Consider two functions $f(n)$ and $g(n)$. In a proof by induction to show that $\forall n \geq 1 : f(n) = g(n)$, what is the induction hypothesis?

    A. $\forall k \geq 1 : f(k) = g(k)$

    B. $f(x) = g(x)$ if $x = 1$

    C. f(x) = g(x) if x = k

    D. $\forall n \geq k : f(n) = g(n)$

    E. $f(n+1) = g(n+1)$

16. (4 points) Consider an undirected graph G whose adjacency matrix is A, i.e. $A[x, y] = 1$ if there is an edge between vertices x and y, and $A[x, y] = 0$ otherwise. If $B = A^2$, what is the value of $B[x, y]$?

    A. 1 if there is a path of length 2 from x to y, and 0 otherwise

    B. 1 if there is a cycle of length 2 between x to y, and 0 otherwise

    C. The number of paths of length 2 from x to y

    D. The number of cycles of length 2 between x and y

    E. None of the above

Short questions

17. (4 points) What should one do in order to increase the page-rank of his web site W, as defined in class?

    A. Add links from W to several "high-profile" pages

    B. Reduce the number of links going from W to other sites

    C. Ask professors to put a link to W from their own web sites

    D. Ask friends who already have a link to W to also include links to "high-profile" pages

    E. None of the above

18. (4 points) Consider a country where coins have value 1, 5, and 6. On which of the following totals will the greedy algorithm fail to obtain an optimal solution?

    A. 8

    B. 9

C. 10

D. 11

E. 12

19. (4 points) Consider the 5-queen problem seen in class, which consists of placing 5 queens on a 5x5 chessboard so that no two queens attack each other. Suppose that the backtracking algorithm seen in class is used to find a solution, and that halfway through its execution, its current partial solution is shown below. When the algorithm encounters its first valid solution, which of the squares on the last row will be occupied by a queen?

	*			
				*
*				
A	B	C	D	E

A. Square A

B. Square B

C. Square C

D. Square D

E. Square E

20. (4 points) When writing Java programs, which of the following statements is/are true?
1. One should always prefer variables of type "double" rather "float", because "double" is more precise.
2. Because they return nothing, methods that have return type "void" should be avoided.
3. A constructor method can only be executed at the beginning of the execution of the program.
4. A program should be broken up in as many methods as possible.

A. 1 and 2 only

B. 1, 2, and 3 only

C. 1, 2, and 4 only

D. 2, 3, and 4 only

E. None of the above

21. (4 points) What is/are the main advantage(s) of public-key cryptography over secret-key cryptography?

A. The sender (Alice) and receiver (Bob) do not have to secretely agree on an encryption key before being able to exchange message.

B. Public-key cryptography is safer

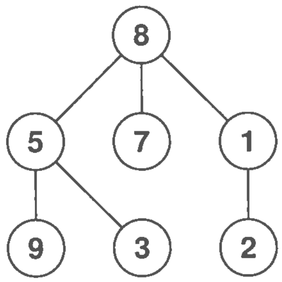
C. Public-key cryptography is faster

D. Message encrypted with public-key cryptography are more compact

E. All of the above

Array-based representation of trees

A ternary tree is a tree such that each internal node has at most 3 children. As illustrated below, ternary trees can be represented with an array.



Graphical representation

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	8	5	7	1	-	9	3	-	-	-	-	2	-

Array-based representation

22. (4 points) Let i be the index of a node n. We want to exchange the left child of n with the right child of n. What are the indices of the nodes we want to exchange?

A. $i \leftrightarrow 2 \cdot i + 2$

B. $i \leftrightarrow 3 \cdot i - 1$

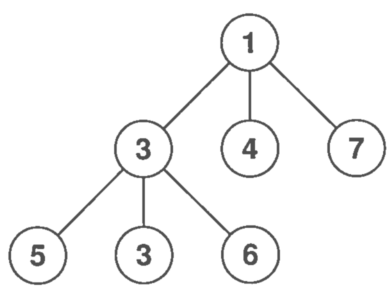
C. $\mathbf{3 \cdot i - 1 \leftrightarrow 3 \cdot i + 1}$

D. $2 \cdot i \leftrightarrow 3 \cdot i + 1$

E. $2 \cdot i \leftrightarrow 2 \cdot i + 1$

Heaps

We can implement ternary heaps just as like the binary heaps. We show in the following figure an example of a min ternary heap.



23. (4 points) What **advantage** does a ternary heap have versus a binary heap?

A. We can store more keys.

B. For the same number of keys, the height of the tree will be smaller.

C. The array-based representation is more compact.

D. In heapify and delete operations, only one comparison is needed to get the minimum of 3 children while swapping for maintaining heap property.

E. None.

24. (4 points) What **disadvantage** does a ternary heap have versus a binary heap?

A. We cannot store has many keys as with a binary tree.

B. For the same number of keys, the ternary tree will be higher.

C. It cannot be represented with an array.

D. In heapify and delete operations, it will take 2 comparisons instead of one to get the minimum of 3 children while swapping for maintaining heap property.

E. None.

Graph problems

Which classical graph problem can be used to appropriately models the following situations?

25. (2 points) A taxi driver is looking for the fastest possible route to take his client to her destination.

A. Eulerian Cycle

B. Hamiltonian Cycle

C. Maximum Clique

D. Graph coloring

E. None of the above

26. (2 points) A police car needs to patrol every street in the city exactly once during their shift.

A. Eulerian Cycle

B. Hamiltonian Cycle

C. Maximum Clique

D. Graph coloring

E. None of the above

27. (2 points) A set of n tasks need to be performed. Several tasks can be performed in parallel, except for certain pairs of tasks that are conflicting and that cannot be performed together. We want to schedule all the tasks to the smallest possible number of time slots.

A. Eulerian Cycle

B. Hamiltonian Cycle

C. Maximum Clique

D. Graph coloring

E. None of the above

Analysis of recursive algorithms

28. (4 points) You have discovered a new algorithm that takes two parameters, m and n, where $m \leq n$. Upon some analysis, you find that the runtime obeys the following formula:

$$T(m,n) = \begin{cases} 2 \cdot T(m-1, n-1) & \text{if } m \geq 1 \\ T(0, n-1) + 1 & \text{if } m = 0 \text{ and } n \geq 1 \\ 1 & \text{if } m = 0 \text{ and } n = 0 \end{cases}$$

Give an explicit formula for $T(m,n)$.

A. $T(m,n) = 2^{n+1} \cdot 2^m$

B. $T(m,n) = 2^m$

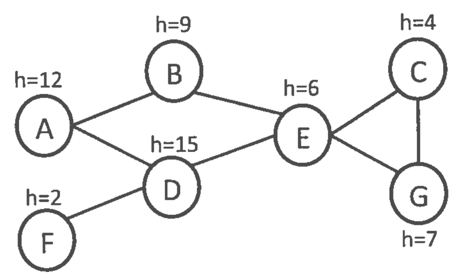
C. $T(m,n) = 2^{m-n+1}$

D. $T(m,n) = 2^m \cdot (n - m + 1)$

E. None of the above

Graphs

Consider the following undirected graph (ignore the values of the h variables for now).



In class, we saw two graph traversal algorithms: depth-first search (DFS) and breadth-first search (BFS). Their pseudocode is given below.

Algorithm 7 DFS(Graph G, vertex v

```
print v
v.setLabel(VISITED)
for all u in v.getNeighbors() do
    if ( u.getLabel() ≠ VISITED ) then
        DFS(G, u)
    end if
end for
```

Algorithm 8 BFS(Graph G, vertex v

```
q = new Queue()
v.setLabel(VISITED)
q.enqueue(v)
while (! q.empty()) do
    w ← q.dequeue()
    print w
    for all u in w.getNeighbors() do
        if (u.getLabel() ≠ VISITED) then
            u.setLabel(VISITED)
            q.enqueue(u)
        end if
    end for
end while
```

29. (4 points) What will be printed when DFS(Graph, A) is executed (i.e. a depth-first search starting at node A of the above graph)? Assume that v.getNeighbors() method returns the neighbors of v in alphabetical order.

A. A B C D E F G

B. A B D E F C G

C. A B E C D G F

D. A B E C D F G

E. None of these answers

30. (4 points) What will be printed by iterativeBFS(Graph, A) (i.e. a breadth-first search starting at node A of the above graph)? Again, assume that v.getNeighbors() method returns the neighbors of v in alphabetical order.

A. A B D E F C G

B. A D B F E G C

C. A B E C G D F

D. A B D E C G F

E. None of these answers

Now, assume that the graph represents the roadmap of a city, with edges representing streets and nodes representing intersections. Each node has a given altitude (h), as shown on the graph. Suppose a major water leak happens at a given intersection. From that node, water flows along the streets (i.e. the edges incident onto that node), but only if the street is downhill (i.e. water flows from node u to node v if and only if (i) node u is flooded and (ii) altitude(u)>altitude(v)). For example, if the leak happens at node A, the water will flow to node B but not D. From B, water will flow to E. From E, water will flow to C but not to G. Water will never reach nodes D, F, and G. We say that nodes A, B, and E undergo a "flowing flood", because they are flooded but water is flowing elsewhere from them. We say that nodes C is under a "standing water flood" because it is flooded but from it water is flowing nowhere.

Complete the algorithm below, which prints out the set of vertices that will be flooded by a leak starting at vertex v, together with their flood status (flowing or standing water). The algorithm should only explore the portion of the graph that gets flooded. For example, when called on vertex A of the graph shown above, it should print (in whichever order you want):

A: flowing flood
B: flowing flood
E: flowing flood
G: standing water flood

Algorithm 9 assessFlood(graph G, vertex v)

```
Input: A vertex v that is flooded in a graph G
Output: Prints the flood status of all nodes that will be affected by the flood.
q ← new MyADT() /* See question below */
v.setLabel(VISITED)
q.insert (v)
while (! q.empty()) do
   w ← q.remove()
   isFlowing ← False;
   for all u in w.getNeighbors() do
      if ( ... LINE1...) then
         isFlowing ← true
         if ( u.getLabel() ≠ VISITED ) then
            u.setLabel(VISITED)
            q.insert(u)
            /* location A*/
         end if
         /* location B*/
      end if
      /* location C*/
   end for
   /* location D*/
end while
```

31. (4 points) In the pseudocode above, we have used the type MyADT. Which type(s) of ADT(s) would be required to make the algorithm work (assuming that insert()/remove() and replaced by their appropriate names for that ADT, e.g. push/pop or enqueue/dequeue)?

A. Queue only

B. Stack only

C. Priority queue only

D. Either a Queue or a Stack

E. None of the above

32. (4 points) Which of the lines below is needed at LINE1 of the algorithm?

A. getAltitude(u) < getAltitude(w)

B. getAltitude(u) > getAltitude(w)

C. getAltitude(u) < getAltitude(w) AND u.getLabel() $\neq$ VISITED

D. getAltitude(u) > getAltitude(w) AND u.getLabel() $\neq$ VISITED

E. None of the above

33. (4 points) What is the appropriate location for the following lines of code?**if** (isFlowing) **then** print w+": flowing flood"

else print w+" standing water flood"

A. Location A

B. Location B

C. Location C

D. Location D

E. None of the above.

Recursive algorithms

34. (4 points) Consider the following *mystery()* recursive algorithm:

Algorithm 10 mystery(int a, int b)

Require: Two non-negative integers a and b.

```
1: if (b = 0) then
2:     return 0
3: else if (b mod 2 = 0) then
4:     return mystery( ____ , ____ )
5: else
6:     return mystery( ____ , ____ ) + a
7: end if
```

Examples of the recursive algorithm's behaviour are below:

input	$\Longrightarrow$	*output*
$mystery(2,\ 50)$	$\Longrightarrow$	100
$mystery(3,\ 35)$	$\Longrightarrow$	105
$mystery(2,\ 25)$	$\Longrightarrow$	50
$mystery(0,\ 9)$	$\Longrightarrow$	0
$mystery(5,\ 5)$	$\Longrightarrow$	25

What arguments should replace the blank statments (i.e., ____) when recursively calling the *mystery()* algorithm (**lines 4 and 6**) to produce the above output?

A. 0 and $b - a$

B. $a + b$ and b / 2

C. a / a and b / b

D. a / 2 and $b + b$

E. a + a and b / 2

Design of recursive algorithms

35. (4 points) A palindrome is a string that reads the same in the forward and reverse directions, such as "laval" or "kayak". Complete the following recursive algorithm that returns `True` if the input string is a palindrome, and `False` otherwise. Given a string s, the substring from positions i to j (inclusively) is written as $s[i : j]$. A character at position i can be accessed by $s[i]$ and the index starts at 0.

Algorithm 11 PalindromeRecursive(String s)

Require: A string s.
if (length(s) ≤ 1) **then**
    return True
else if $s[0] = s[\text{length}(s) - 1]$ **then**
    return PalindromeRecursive($\cdots$)
end if

A. $s[0 : \text{length}(s)]$

B. $s[1 : \text{length}(s)]$

C. $s[0 : \text{length}(s) - 1]$

D. $s[1 : \text{length}(s) - 1]$

E. None of the above.

Recursive algorithms

Consider the "Tribonacci" sequence $T_0, T_1, T_2, \ldots$ defined as follows:

$$T_n = \begin{cases} n & \text{if } n \leq 2 \\ T_{n-3} + T_{n-2} + T_{n-1} & \text{if } n \geq 3 \end{cases}$$

36. (4 points) What is the value of T_6 ?

A. 6

B. 11

C. 20

D. 37

E. None of the above

37. (4 points) The following recursive algorithm computes T_n:

Algorithm 12 Trib(int n)

if (n < 3) **then**
    return n
else
    return Trib(n-3) + Trib(n-2) + Trib(n-1)
end if

Let $X(n)$ be the total number of times the Trib() method will be called during the execution of Trib(n) (including the call to Trib(n) itself). For example, $X(0) = 1, X(3) = 4, X(4) = 7$.

Write a recurrence for $X(n)$.

A. $X(n) = \begin{cases} 1 & \text{if } n < 3 \\ X(n-1) + X(n-2) + X(n-3) & \text{if } n \geq 3 \end{cases}$

B. $X(n) = \begin{cases} n & \text{if } n < 3 \\ X(n-1) + X(n-2) + X(n-3) & \text{if } n \geq 3 \end{cases}$

C. $X(n) = \begin{cases} 1 & \text{if } n < 3 \\ X(n-1) * X(n-2) * X(n-3) & \text{if } n \geq 3 \end{cases}$

D. $X(n) = \begin{cases} n & \text{if } n < 3 \\ X(n-1) * X(n-2) * X(n-3) & \text{if } n \geq 3 \end{cases}$

E. None of the above

Sorting

Suppose that while your computer is sorting an array of integers, its memory is struck by a cosmic ray that changes exactly one of the keys to something completely different.

38. (4 points) What is the **worst-case** possibility if you are using **MergeSort**?

A. The final array will be fully sorted except exactly one key.

B. The final array will have just one or two keys out of place.

C. The final array will consist of two separate sorted subsets, one following the other, plus perhaps one or two additional keys out of place.

D. The final array will not even be close to sorted.

E. The algorithm will stop and the final array will be incomplete.

MergeSort algorithm

The following algorithms are variants of the MergeSort algorithm. For each variant, indicate the statement that best describes it (note: define n = stop - start + 1).

39. (4 points) .

Algorithm 13 MergeSort1(int A[], int start, int stop)

if (start<stop) **then**
  mid ← (start+stop)/2
  merge(A, start, mid, stop)
  mergeSort1(A,start,mid)
  mergeSort1(A, mid+1, stop)
end if

A. The algorithm will always produce the correct answer and runs in time O(n log(n))

B. The algorithm will always produce the correct answer but does not runs in time O(n log(n))

C. The algorithm will not always produce the cor-rect answer and runs in time O(n log(n))

D. The algorithm will not always produce the correct answer and does not runs in time O(n log(n))

E. The algorithm produce an infinite recursion

40. (4 points) .

Algorithm 14 MergeSort2(int A[], int start, int stop)

if (start<stop) **then**
  mid ← $\lfloor (start + stop)/2 \rfloor$
  MergeSort2(A,start,mid)
  MergeSort2(A, mid+1, stop)
  MergeSort2(A,start,mid)
  merge(A, start, mid, stop)
end if

A. The algorithm will always produce the correct answer and runs in time O(n log(n))

B. The algorithm will always produce the correct answer but does not runs in time O(n log(n))

C. The algorithm will not always produce the correct answer and runs in time O(n log(n))

D. The algorithm will not always produce the correct answer and does not runs in time O(n log(n))

E. The algorithm produce an infinite recursion

41. (4 points) .

Algorithm 15 MergeSort3(int A[], int start, int stop)

if (start<stop) **then**
  mid ← start
  MergeSort3(A,start,mid)
  MergeSort3(A, mid+1, stop)
  merge(A, start, mid, stop)
end if

A. The algorithm will always produce the correct answer and runs in time O(n log(n))

B. The algorithm will always produce the correct answer but does not runs in time O(n log(n))

C. The algorithm will not always produce the correct answer and runs in time O(n log(n))

D. The algorithm will not always produce the correct answer and does not runs in time O(n log(n))

E. The algorithm produce an infinite recursion

Big-O notation

42. (4 points) Consider the function $f(n) = \log(n!)$ and $g(n) = n\log(n^2)$. Which of the following statements holds?
    A. f(n) is O(g(n)) and g(n) is O(f(n))
    B. $f(n)$ is $O(g(n))$ but $g(n)$ is not $O(f(n))$
    C. $f(n)$ is not $O(g(n))$ but $g(n)$ is $O(f(n))$
    D. $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O(f(n))$
    E. None of the above

43. (4 points) Consider the function $f(n) = n\log(n) + n + 1$ and $g(n) = 2n$. Which of the following statements holds?
    A. $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$
    B. $f(n)$ is $O(g(n))$ but $g(n)$ is not $O(f(n))$
    C. f(n) is not O(g(n)) but g(n) is O(f(n))
    D. $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O(f(n))$
    E. None of the above

44. (4 points) Consider two non-negative functions $f(n)$ and $g(n)$ such that $f(n)$ is $O(g(n))$. Which of the following statements may **not** hold?
    A. $\log(f(n))$ is $O(\log(g(n)))$
    B. $\log(f(n))$ is $O(g(n))$
    C. $f(n)$ is $O(2^{g(n)})$
    D. $\log(f(n))$ is $O(2^{g(n)})$
    E. 2f(n) is O(2g(n))

45. (4 points) What is the Big O notation for the best case and worst case running time of the SelectionSort algorithm seen in class?
    A. best case: $O(1)$, worst case: $O(n)$
    B. best case: $O(1)$, worst case: $O(n\log(n))$
    C. best case: $O(n)$, worst case: $O(n\log(n))$
    D. best case: $O(n\log(n))$, worst case: $O(n^2)$
    E. best case: O(n2), worst case: O(n2)

Faculty of Science
FINAL EXAMINATION

COMPUTER SCIENCE COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Examiner: Prof. Michael Langer
Associate Examiner: Prof. Paul Kry

Dec. 13, 2012
9 A.M. – 12 P.M.

STUDENT NAME: ______________________ **ID:** __________________

Instructions:

The exam is 14 pages, including this cover page.

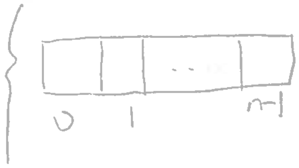
There are 10 questions, worth a total of 60 points.

Answer all questions on the exam question sheet.

An extra page is included at the end for sketching out solutions.

No calculators, notes, or books are allowed.

Question	Points	Grade
1	4	
2	4	
3	4	
4	6	
5	6	
6	7	
7	10	
8	7	
9	6	
10	6	
Total	60	

elementData 

Remove last O(1)

else O(n)

1. **(4 points)**

(a) Fill in the missing code in the indicated region /*..*/ for the following method in a Java ArrayList<T> class implementation. The method should remove and return the list element at index i.
State any assumptions you make about the private fields of the class.

instance method of AL class

```
public  T  remove(int index){
    T  tmp;
    if ((index >= 0) && (index < size)){
      /*
      tmp = elementData[index];   // assume elementData as a field
      for (int i = index + 1 , i < size , i++) {
          elementData[i-1] = elementData[i] ; // shifting
      }
      size-- ;
      return tmp ;
      */
    }
    else {
        System.out.println("array index out of bounds");
        return null;
    }
}
```

(b) What is the $O(\ )$ and $\Omega(\ )$ bound for this method? Briefly justify your answer.

list → "Bob" → "Bobby" → "Bob"

list 2 → "Bobby"

1st outer loop: list 1 → "Bob" → null

"Bobby" → "Jack" → null

2nd outer loop: list 1

reference point

2. **(4 points)**

Consider the following Java method.

input

```
LinkedList<String>  mysteryMethod( LinkedList<String>  list ){
    String  s1,s2;
    LinkedList<String>  list1 = new LinkedList<String>();
    LinkedList<String>  list2;
    while (!list.isEmpty()){
       s1 = list.removeFirst();
       list1.add(s1);
       list2 = new LinkedList<String>();
       while (!list.isEmpty()){
          s2 = list.removeFirst();
          if (!s1.equals(s2)){
             list2.add(s2);
          }
       }
       list = list2;
    }
    return list1;
}
```

(a) The method takes a linked list of strings as input. What does the method compute ?

Removes

(b) Give the $O(\ )$ bound on the number of calls to the `equals()` method, that is, the number of `String` comparisons required by the method. Justify your answer.

(c) Same as (b), but give the $\Omega(\ )$ bound. Justify your answer.

3. **(4 points)**

(a) Give the formal definition for "$t(n)$ is $O(g(n))$".

$$t(n) \begin{cases} 1 & \text{when } n\%2 \neq 0 \\ 1+2n & n\%2 = 0 \end{cases}$$

(b) Let $t(n)$ be such that $t(n) = 1$ when n is odd and $t(n) = 1 + 2n$ when n is even, *i.e.*

n	1	2	3	4	5	6	7	8	...
t(n)	1	5	1	9	1	13	1	17	...

Underline the asymptotic bounds below for which we can say "$t(n)$ is _____".

$O(1)$ $O(\log n)$ $O(n)$ $O(n^2)$ $\Omega(1)$ $\Omega(\log n)$ $\Omega(n)$ $\Omega(n^2)$

You do *not* need to justify your answers.

$2^3 = 2^2 \cdot 2 = 2 \cdot 2 \cdot 2$

4. **(6 points)**

(a) Below left is the recursive algorithm for `power(x,n)`. Fill in the missing code on the right of a non-recursive version of the same algorithm which uses a stack instead of recursion.

RECURSIVE

pop

```
power(x,n) {
  if (n == 0)
    return 1      recursive
  else{                  push
    tmp = power(x, n/2)
    if  (n % 2 == 0)
      return tmp*tmp
    else
      return tmp*tmp*x
  }
}
```

NON-RECURSIVE

push

pop

```
power(x,n) {
   tmp = 1
   s = empty stack
   while  ( n > 0 ){
        s. push (n% 2);
        n /=2
   }
   while ( ! s.isEmpty()  ){
        oddEven = s.pop() ;
        if ( oddEven == 1) {
             tmp = tmp*tmp*x ;
        } else {
             tmp= tmp* tmp;
   }   }
   return tmp;
}
```

(b) For the recursive algorithm, write a recurrence relation for the *upper bound on the number of multiplications* needed to compute `power(x,n)`, and solve the recurrence *i.e. solve for the upper bound*. Use the other side of the page, if necessary.

① $t(n) = t(\frac{n}{2}) + \cancel{t(n)} + KC$ (K=1) (K=2)

$= t(\frac{n}{2}) + 2$

$= 2 + 2 + t(\frac{n}{4})$

$= 2k + t(\frac{n}{n})$

$= 2 \cdot \log(n) + t(1)$

5. **(6 points)**

Consider the binary tree:

```
        h
      /   \
    p       m
      \    / \
       w  f   y
      /
    d
```

(a) What is the order of nodes visited in a *pre-order* traversal? ROOT LC RC

h p w d m f y

(b) What is the order of nodes visited in an *in-order* traversal? LC ROOT RC

p d w h f w y

(c) What is the order of nodes visited in a *post-order* traversal? LC RC ROOT

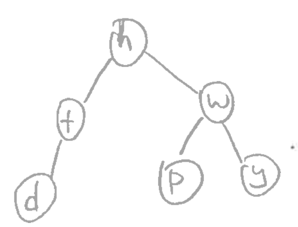
d w f f y m h

(d) Transform this binary tree into a *binary search tree* of height 2, defined by the natural ordering on characters.

d f h m p w y

Unique

(e) Show the result of removing `m` from your binary search tree.



node is root

② add

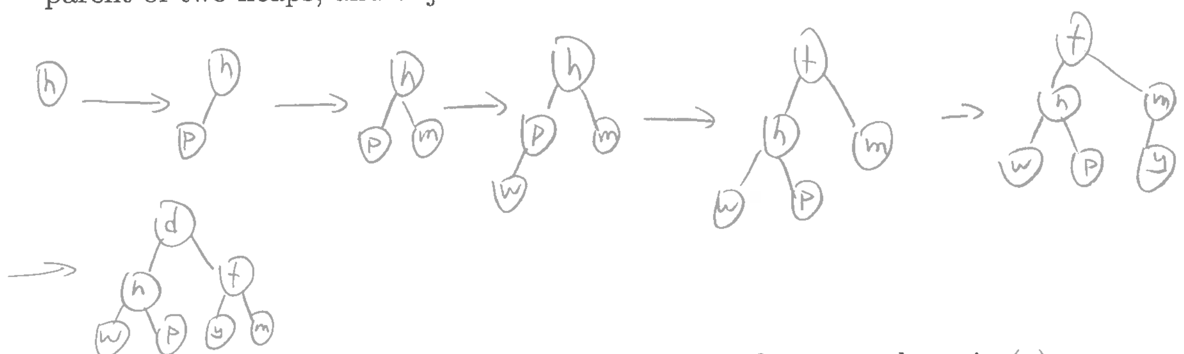
③ bubble

6. **(7 points)**

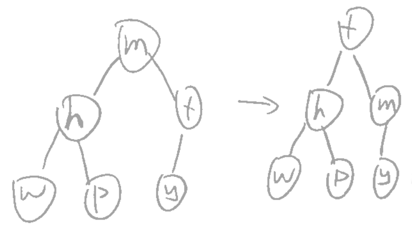
(a) Consider an list of keys (h, p, m, w, f, y, d). Assuming this is the initial ordering of the keys in an array implementation, use the fast `buildHeap()` algorithm, i.e.

```
for (k = n/2; k >= 0; k--)
    downHeap( a, n, k  )
```

to build a (min)-heap from these keys. Recall that `downHeap()` assumes node **k** is the parent of two heaps, and adjusts the tree so that node **k** becomes the root of a heap.



(b) Show the result of removing the minimum element from your heap in (a).



(c) Give $O()$ and $\Omega()$ bounds for the following heap operations, as a function of the number n of keys in the heap. Briefly justify your answers.

- `removeMin()`, which returns the minimum key;

  find Min : $O(1)$

  Remove Min : $O(\log(n))$

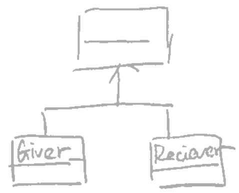
  $\Omega(1)$: no bubbling down

  $O(\log(n))$ bubble all the way down

- `remove(key)`, which removes a given **key** if it is present in the heap. (If there are multiple copies of the key then this method just removes one copy.)

7. **(10 points)**

Suppose the classes below are all in the same package. See questions on next page.




```
public class Sharer{
    int    sum;
    String  ID;
    Sharer other;

    public Sharer(String ID, int sum){ this.ID = ID;  this.sum = sum; }

    //  give half,  keep half
    void  share(int n){  other.sum += n/2;  this.sum += n - n/2;  }

    public String toString(){ return ID + " " + sum + " "; }
}

class Giver extends Sharer{

    public Giver(String ID, int sum) { super(ID, sum); }

    void share(int n) {   other.sum += n;  this.sum  -= n;  }
}

class Taker extends Sharer{

    public Taker(String ID, int sum) {  super(ID, sum);  }

    void share(int n) {   other.sum -= n;  this.sum  += n;  }
}

public class Test {

   public static void main(String[] args) {
        Sharer a = new Giver("Geoff", 10);
        Sharer b = new Taker("Tina",  7);
        Sharer c = new Taker("Ted",  15);

        a.other = b;   b.other = c;   c.other = a;
        a.share(2);    b.share(4);    c.share(7);

        System.out.println( a.toString() +  b.toString() + c.toString());
   }
}
```

a. sum = 8
b. sum = 9

b. sum = 13
c. sum = 11

c. sum = 18
a. sum = 1

Geoff 1   Tina 13   Ted 18.

(a) What is the output when the `Test` class is run?

(b) Suppose we were re-define the visibility of the three fields of `Sharer` to be `private` so that the `Giver` and `Taker` no longer have access to these fields.

Rewrite the methods of `Sharer`, `Giver`, and `Taker` to be consistent with this new definition. *Only re-write methods that change.* You can rewrite the methods next to the code on the previous page and/or below here.

8. **(7 points)**

Consider the GUI that is defined by the following class.

```
class MyPanel extends JPanel{

   int fontSize = 12;
   JButton button;
   JLabel  myLabel;

   MyPanel(){
      button = new JButton();
      ButtonListener listener = new ButtonListener();
      button.addActionListener( listener );
      myLabel = new JLabel("my button");
      button.add(myLabel);
      setLayout( new FlowLayout());
      add(button);
   }

   private class ButtonListener implements ActionListener{
      public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button){
           myLabel.setFont( new Font("Times", Font.BOLD, fontSize++ ));
        }
      }
   }
}
```

(a) Briefly describe the interface: What does the user see? What does the user do, and what does the user see as the result ?

(b) What are the objects produced by this GUI and by the user's actions?

(c) Summarize the steps that occurs internally in the program, that lead from the user's action to the change that the user sees on the screen? In particular, your answer should discuss the relationships between the objects listed in (b).

9. **(6 points)**

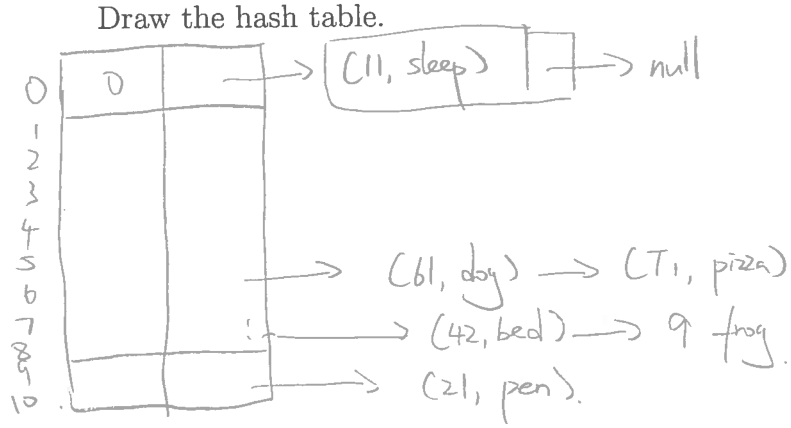
(a) Consider a hash table with (key, value) pairs,

$$(11,\ \texttt{sleep}),\quad (42,\texttt{bed}),\quad (9,\texttt{frog}),\quad (61,\texttt{dog}),\quad (17,\texttt{pizza}),\quad (21,\texttt{pen})$$

and hash function

$$h(k) = k \bmod 11.$$

Draw the hash table.



(b) Consider a hash table with capacity of m buckets, and n entries *i.e.* n (key, value) pairs.

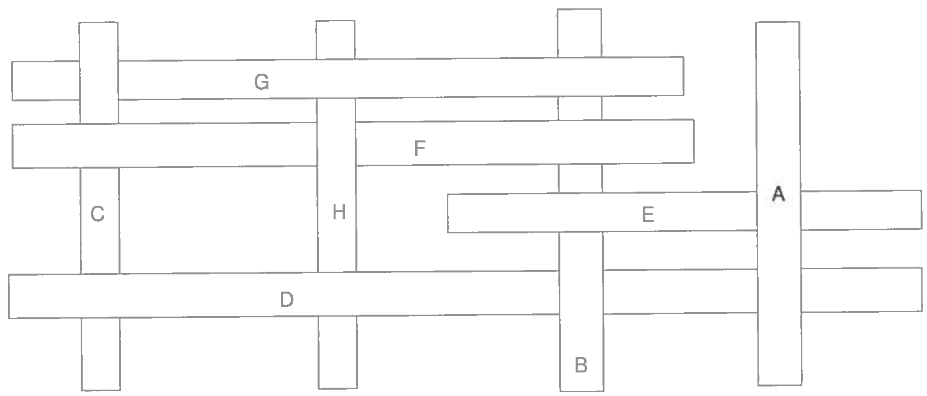
- Give a $O()$ bound for the method `contains(K key)` which examines whether there is an entry in the hash table with that key. Justify your answer, and state your assumptions.

low load factor

- Give a $O()$ bound for the method `contains(V value)` which examines whether there is a particular value in the hash table. Justify your answer, and state your assumptions.

10. **(6 points)**

The figure below shows a set of rectangles whose overlap relationships can be represented using a directed graph, as follows. Each rectangle is represented in the graph by one vertex, and there is a directed edge in the graph whenever part of one rectangle lies on top of another rectangle. For the example below, there is an edge from `G` to `H`, but not from `H` to `G` and there are no edges between `A` and `B` since these rectangles do not overlap at all.



(a) Give the adjacency list for this graph, such that *the vertices are ordered alphabetically.* Note that this ordering implies that there is a unique answer.

A → D,E.
B → D
C → G
D → C,H
E → B
F → B,C
G → B,H
H → F

(b) Using the adjacency list in (a), give the ordering of vertices visited in a *breadth first* traversal of the graph, starting from vertex `A`.

(c) Using (a), give the ordering of vertices visited in a *pre-order (depth first)* traversal, again starting from `A`.

ADCGBHFE

Faculty of Science
FINAL EXAMINATION

COMPUTER SCIENCE COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Examiner:	Prof. Michael Langer	April 27, 2010
Associate Examiner:	Mr. Joseph Vybihal	9 A.M. – 12 P.M.

Instructions:

The exam is 6 pages, including this cover page.

There are 9 questions, worth a total of 30 points.

Answer all questions in the exam book. You may keep the exam question sheet.

No calculators, notes, or books are allowed.

1. **(3 points)**

   (a) Convert the decimal number 231 to binary.

   (b) What is the value of the hash function,

   $$h(i) = i \bmod 16$$

   for $i = 231$? You may express your answer in binary.

   (c) How many multiplications are used to compute 13^{231} using the recursive method discussed in class *i.e.* power(13,231) ?
   Note: The right side of "$13^{231} = 13^{115} \times 13^{115} \times 13$" is two multiplications.

2. **(2 points)**

State the formal definition of "$f(n)$ is $O(g(n))$" and use it to prove $f(n)$ is $O(g(n))$ for the following:

$$f(n) = (n + 237)^3 + 4n \log n + 3$$

$$g(n) = n^3.$$

Hint: You may assume $\log n \leq n$ for all n.

3. **(2 points)**

   (a) Show the final state of a *queue*, after the following sequence of operations is finished:

   `add(a), add(b), add(c), add(d), remove(), add(e), add(f), remove(), remove(), add(g), add(h)`

   Assume the queue is initially empty. Be sure to indicate the front of the queue.

   (b) Answer the same question as (a), but now use a *stack* instead of a queue. Here, indicate the top of the stack.

4. **(3 points)**

   State *without proof* the solution of the following recurrence relations.

   For (a)-(d), assume $t(0) = 1$. For (e),(f), assume $t(1) = 1$ and n is a power of 2.

   (a) $t(n) = 1 + t(n-1)$
   (b) $t(n) = c + t(n-1)$
   (c) $t(n) = n + t(n-1)$
   (d) $t(n) = c\ t(n-1)$
   (e) $t(n) = c + t(\frac{n}{2})$
   (f) $t(n) = n + 2\ t(\frac{n}{2})$

5. **(2 points)**

   Consider a mergesort algorithm that partitions a set of n comparable elements into three equal-size subsets, sorts each of them, and then merges the three sorted subsets. You may assume n is a power of 3.

   (a) Give a recurrence relation that describes the number of operations used for this algorithm as a function of n.

   (b) Solve this recurrence relation. Be sure to show the steps used to obtain your solution.

6. **(3 points)**

   Consider the following array of characters:

   $$\texttt{minHeap} = [\texttt{b}, \texttt{c}, \texttt{m}, \texttt{h}, \texttt{d}, \texttt{p}, \texttt{w}]$$

   which defines a min-heap.

   (a) What is the result of applying the heap's `removeMin()` operation? Your answer must be a new array with six elements.

   (b) Give a binary search tree of height 2 that contains the original characters of `minHeap`, *i.e.* before the operation in (a).

   (c) Apply the binary search tree operation `remove(h)` to your tree in (b).

7. **(5 points)**

(a) Using pseudocode, write a *recursive* algorithm `findRoot(node)`, whose input is a node in a tree and whose output is the root node of the tree. For example, in the trees below, the input could be any node and the output would always be node '`a`'.

Assume that each node in the tree has a field `parent`, and that the input node is valid, *i.e.* it references a node in the tree.

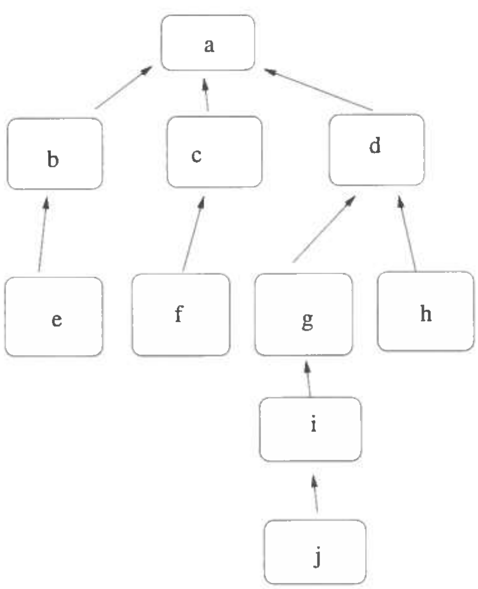
(b) Consider an algorithm that transforms a tree such that all non-root nodes in the path from the input `node` to the tree `root` will have depth 1 in the transformed tree. The algorithm is called using:

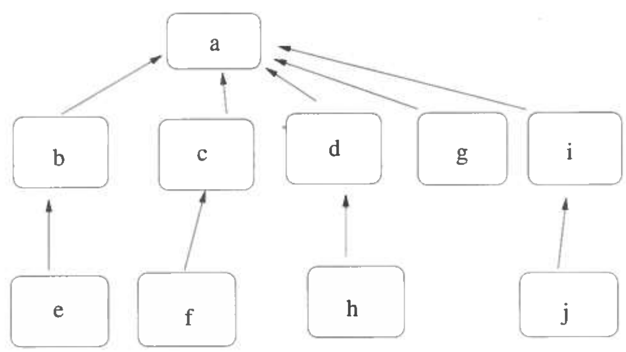
`flatten(node, root)` .

An example of the transformation is shown below (left to right). For this example, what is the input `node` that explains the transformation?

(c) Using pseudocode, write a *recursive* algorithm `flatten( .. )`. Assume that the inputs `node` and `root` are valid.

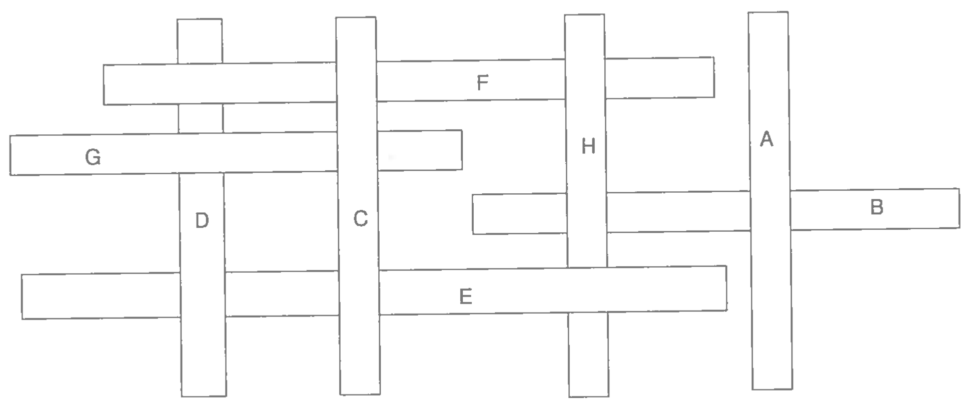
(d) What might be the advantage of calling `flatten` after you call `findRoot`?





8. **(3 points)**

The figure below shows a set of rectangles whose overlap relationships can be represented using a directed graph. Each rectangle is represented in the graph by one vertex, and whenever one rectangle overlaps another rectangle, there is a directed edge. For the example below, the graph would contain the edges $\{(\texttt{C}, \texttt{G}),\ (\texttt{C}, \texttt{E}), \ldots, etc\}$.



(a) Give the adjacency list for this graph, such that *the vertices are ordered alphabetically.*

(b) Give the ordering of vertices visited in a *breadth first* traversal of the graph, starting from vertex C.
NOTE: In this question and the next, there is only one answer since you must order the vertices alphabetically.

(c) Give the ordering of vertices visited in a *depth first* traversal, again starting from C.

9. **(7 points)**

   (a) What is the difference between *overriding* and *overloading* ?

   (b) When you write an `equals()` and `hashCode()` method for a class, what (if any) are the required relationships between the outputs of these method?

Next, consider the Java code below. This code does not do anything meaningful. Rather, it is used to test your understanding of inheritance and polymorphism.

Which (if any) of the instructions `(c)-(g)` of `Test` generate compiler errors?

What is the output of the program after removing those lines (if any) that cause compilation errors? *Be sure to mark which instructions produce which output.*

```
public class A {
   public  int   n  = 3;

   A( ){ System.out.println("A"); }           //  constructor
   public void  foo() { System.out.println( n ); }
}

public class B extends A{
   public  int     n;

   B() { }                                      //  constructor
   B(int m){                                    //  constructor
      System.out.println( "B" );
      this.n = m;
   }
   int  foo(int n) {
           System.out.println(this.n) ;
           return n;
   }
}

public class Test {

    public static void main(String[] args) {
       int n = 2;
       A   a = new A();                         //          (c)
       a.foo();                                 //          (d)
       a.foo( 7 );                              //          (e)
       B   b  =  new B( 11 );                   //          (f)
       n = b.foo( 4 );                          //          (g)
    }
}
```

Faculty of Science
FINAL EXAMINATION

COMPUTER SCIENCE COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Examiner:	Prof. Michael Langer	April 20, 2009
Associate Examiner:	Mr. Joseph Vybihal	9 A.M. – 12 P.M.

Instructions:

Read all questions before you start. The number of points per question is unrelated to the difficulty of the questions, so use your time wisely.

The exam is five pages (doublesided), including this cover page.

There are seven questions, worth a total of 30 points.

Answer all questions in the exam book. You may keep the exam question sheet.

No calculators, notes, or books are allowed.

1. **(3 points)**

Consider the following binary numbers, which are each between 0 and 1:

$$p_1 = .0011, \quad p_2 = .0101, \quad p_3 = .0001, \quad p_4 = .0101 \quad , p_5 = .001$$

Define

$$F_i = \sum_{k=1}^{i} p_k \ , \quad i = 1, 2, 3, 4, 5.$$

(a) Write each F_i as a binary number.

(b) Draw a complete binary tree of height $h = 4$ and *label the leaves* with labels from $\{1, \ldots, 5\}$ increasing left to right, such that p_i is the fraction of leaves with label i.

2. **(2 points)**

(a) Describe what the following *recursive method* does. The method has one parameter which is a reference to a node in a list.

```
myPrintMethod(ListNode node){
   if (node.prev != null){
      myPrintMethod( node.prev );
   }
   System.out.println("  " + node.toString());
}
```

(b) Give and solve a recurrence relation for the runtime of this method in terms of the number n of nodes in the list.

3. **(5 points)**

State the formal definitions of "$f(n)$ is $O(g(n))$", and "$f(n)$ is *not* $O(g(n))$".
Use these formal definitions to prove the following:

(a) $3n + 8$ is $O(n)$.

(b) $3n + 8$ is *not* $O(1)$.

(c) $6n$ is $O(2^n)$. (You *must* prove this one by induction.)

You will lose marks if your definitions or proofs take the limit as $n \to \infty$.

4. **(5 points)**

What is the $O(\,)$ for the runtime of the following operations, in terms of the number of elements n in the data structure ? If you are unsure, then state any assumptions you make.

Note: 'get' returns an element but does not 'remove' it.

(a) Get the i^{th} element from an array.

(b) Insert a new element into a sorted array.

(c) Add an element to the front of a linked list.

(d) Get the i^{th} element from a linked list.

(e) Remove the i^{th} element from a linked list.

(f) Add an element to a minHeap.

(g) Get the minimum element from a minHeap.

(h) Get the maximum element from a minHeap.

For (f)-(h), assume the heap is represented using an array.

5. **(3 points)**

(a) Show the final contents a *queue*, after the following sequence of operations have been performed. Be sure to indicate the front and back.

```
initialize empty queue
enqueue(a)
enqueue(b)
enqueue(c)
dequeue()
enqueue(d)
dequeue()
enqueue(e)
enqueue(f)
dequeue()
enqueue(g)
enqueue(h)
```

(b) Suppose we implement the queue as a circular queue, using an *array* that is initialized with size $m = 4$. Show the contents of the array after *each* of the above operations have been performed. Note that the number of `enqueue` operations is five more than the number of `dequeue` operations, so it will be necessary at some point to increase the size of the array.

6. **(4 points)**

(a) What is the maximum number of nodes in a *binary tree* of height h ?

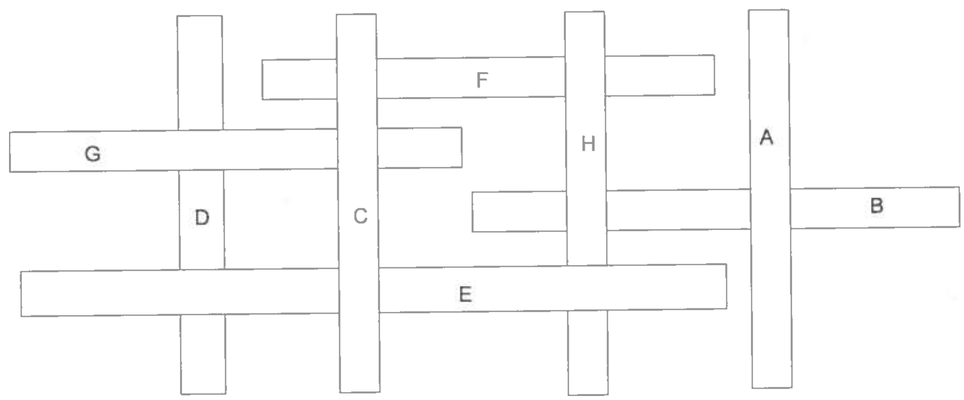
(b) What is the sum of the depths of all nodes in a *binary tree* of height h, assuming all levels of the tree are full? It is sufficient to write a summation expression here. You are not required to evaluate this summation.

(c) Give a *recursive* algorithm `NumNodes(node)` for computing the number of nodes in a tree whose root node is the argument. The tree is not necessarily a binary tree.

(d) Give a *recursive* algorithm `SumDepths( ... )` for computing the sum of the depths of all nodes in a rooted tree. You will need to choose the argument(s) appropriately. As in (c), the tree is not necessarily binary.

7. **(8 points)**

The figure below shows a set of rectangles whose overlap relationships can be represented with a directed graph. Each rectangle is represented by one vertex, and there is a directed edge whenever one rectangle overlaps another rectangle. For example, there is an edge (A,B).



(a) Give the adjacency list for this graph. *The vertices must be ordered alphabetically.*

(b) Draw the graph, labelling the vertices A,B,C,D,E,F,G,H.

(c) Give the ordering of vertices visited in a *breadth first search* traversal of the graph, starting from vertex C. *Show the BFS tree.*

NOTE: In this question and the next, there is only one answer since you must order the vertices alphabetically.

(d) Give the ordering of vertices visited in a *depth first search* traversal, starting from vertex C. *Show the DFS tree.*

(e) Give a valid topological sorting of the vertices in this graph.

(f) Give a *new* example having four rectangles I,J,K,L, such that corresponding graph contains a *cycle*. Draw the rectangles and the graph.