92.5

CSCI-140/242 Term 20175	Computer Science AP/Transfers	Final Exam December 14, 2017

Final Exam

Full Name _____

RIT User N _____

Circle your sect

		r	Exam Location
		Ghunaim	GOL-1435
		ut	GOL-1435
3	4:35-6:35pm	Sean Strout	GOL-1435
4	10:10am-12:10pm	James Heliotis	GOL-1445

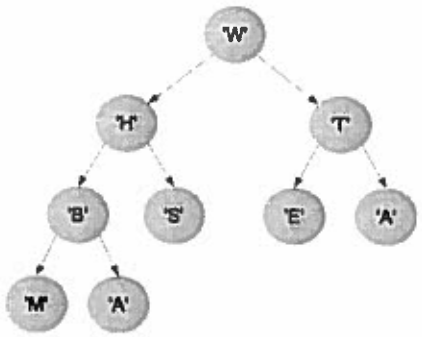
Notes:

- The exam is worth 20% of the course grade.
- The exam has 94 points.
- The exam has 39 problems. .
- The exam has 30 pages.
- The exam period is 2 hours.
- The exam is closed book and closed notes.
- The exam is closed electronics (no laptops, netbooks, OLPCs, tablets, iPads, calculators, cellular phones, iPhones, Nexi, iPods, Zunes, Kindles, Nooks, . . .)

5

6. (6 points)

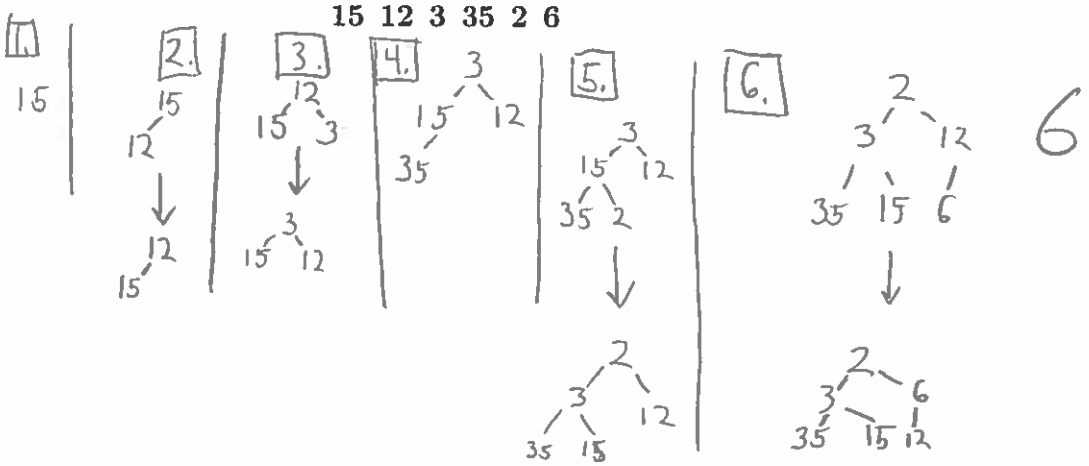
- (2 points) Fill in the array that would represent the complete binary tree shown here.



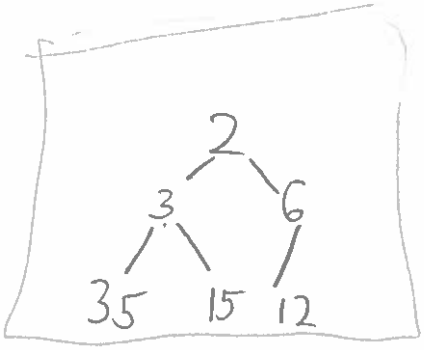
'W'	'H'	'T'	'B'	'S'	'E'	'A'	'M'	'A'

- (2 points) Draw the complete-binary-tree min-heap that is constructed by performing six **add** operations with the following numbers, starting with an initially empty heap. For partial credit, you must show your work.

15 12 3 35 2 6

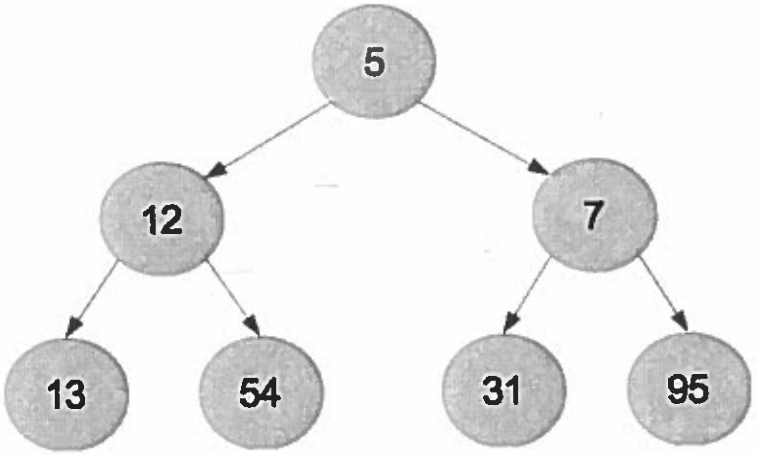


6

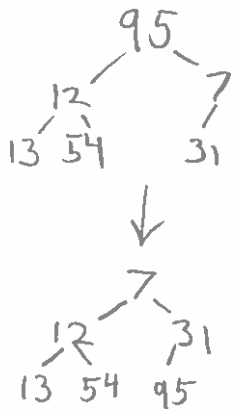


(continued)

- (2 points) Draw the complete-binary-tree min-heap that is generated by performing one **remove** operation, using the following heap.



(5)

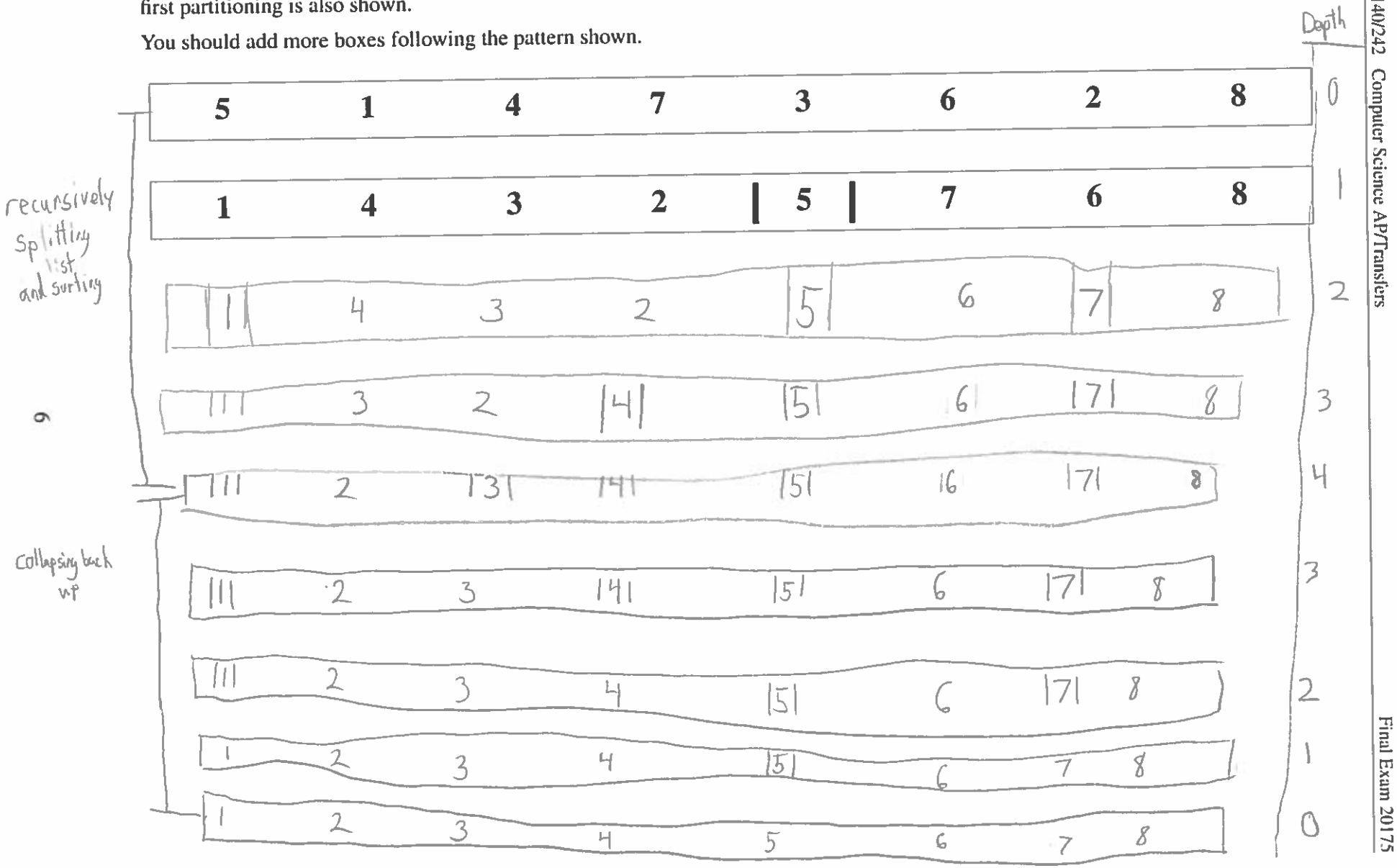


7. (3 points) Trace the execution of **Merge Sort** on the data shown. The final merge is also shown. In the diagram, "Depth n" indicates the depth of the recursive calls.

10	**2**	**8**	**14**	**6**	**12**	**4**	**16**	Depth 0
10	2	8	14	6	12	4	16	Depth 1
10	2	8	14	6	12	4	16	Depth 2
10	2	8	14	6	12	4	16	Depth 3
2	10	8	14	6	12	4	16	Depth 2
2	8	10	14	4	6	12	16	Depth 1
2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	Depth 0

8. (3 points) Trace the execution of **Quick Sort** on the data shown, always using the first value in a list as the pivot for partitioning. The first partitioning is also shown.

   You should add more boxes following the pattern shown.



16. (4 points) Consider the following hash function for strings.

```
def hashFunc( strVal ):
  return numCode( strVal[0] )
```

The `numCode()` function returns the ordinal position of its single-character argument within the English alphabet. For example `numCode('a')` is 0 and `numCode('z')` is 25. (The behavior of `numCode` on multi-character arguments or on single-character arguments that are not characters of the English alphabet is unspecifed, but not relevant to this problem.)

- (2 points) If the value of `tablesize` is 5, then what is `( hashFunc( "batman" ) % tablesize )`?

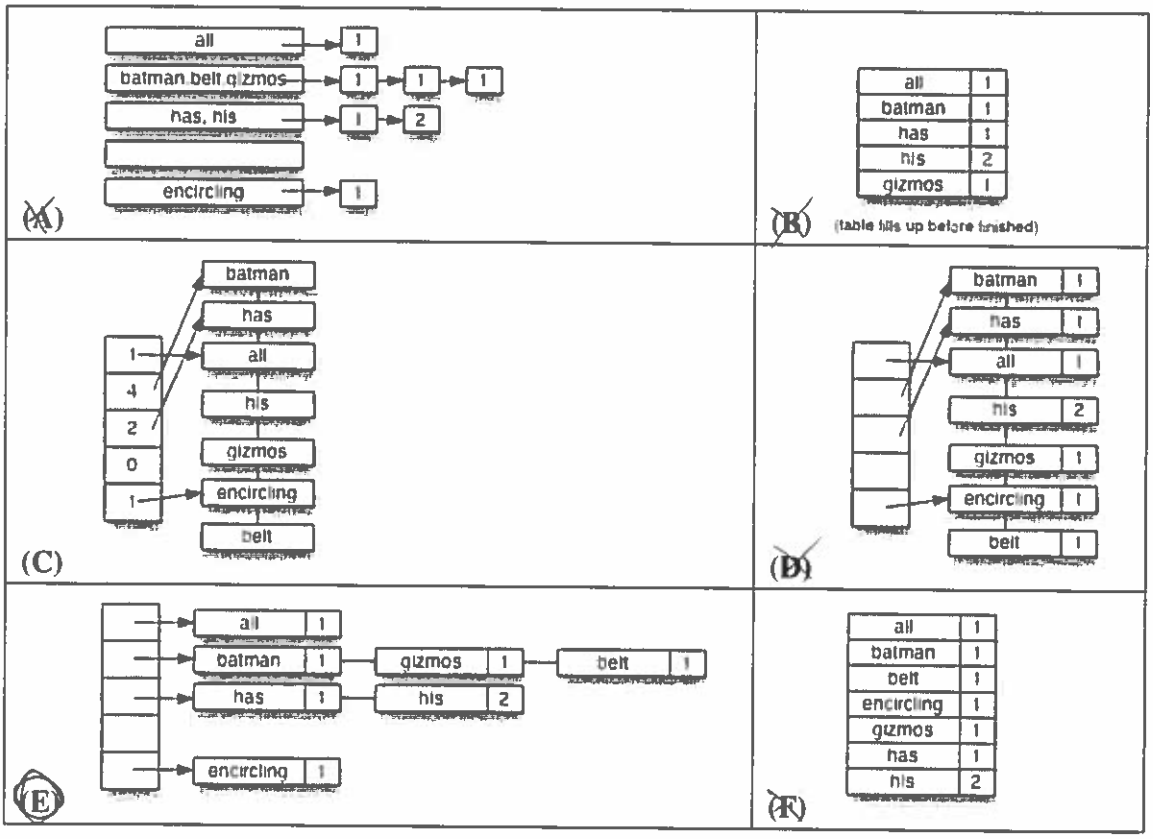
  1 % 5 = 1

  [1]

- (2 points) Consider using a hash table to count the occurrences of words in a text document; the keys are words (strings) and a key's value is the number of occurrences of the word. Circle the figure below that represents a hash table with chaining using an array of size 5, where the following words have been inserted into the hash table:

  `"batman" "has" "all" "his" "gizmos" "encircling" "his" "belt"`

4



17. (2 points) Consider the following list of numbers.

5 2 9 1 4 6 8

Circle the choice that correctly completes the following statement. The progression that insertion sort would produce while sorting this list of numbers in ascending order is:

(A)

5	2	9	1	4	6	8
9	2	5	1	4	6	8
9	2	5	1	4	6	8
9	4	5	1	2	6	8
9	4	6	1	2	5	8
9	4	8	1	2	5	6
8	4	6	1	2	5	9
6	4	5	1	2	8	9
5	4	2	1	6	8	9
4	1	2	5	6	8	9
2	1	4	5	6	8	9
1	2	4	5	6	8	9

(B)

5	2	9	1	4	6	8
2	5	9	1	4	6	8
2	5	9	1	4	6	8
1	2	5	9	4	6	8
1	2	4	5	9	6	8
1	2	4	5	6	9	8
1	2	4	5	6	8	9

(C)

5	2	9	1	4	6	8
2	5	1	4	6	8	9
2	1	4	5	6	8	9
1	2	4	5	6	8	9
1	2	4	5	6	8	9

(D)

5	2	9	1	4	6	8
1	2	9	5	4	6	8
1	2	9	5	4	6	8
1	2	4	5	9	6	8
1	2	4	5	9	6	8
1	2	4	5	6	9	8
1	2	4	5	6	8	9

18. (2 points) Consider the following Python code.

```
def mystery(tree):    # tree is a binary tree
    if tree == None:
        return 0
    elif tree.left == None and tree.right == None:
        return 1
    else:
        return mystery(tree.left) + mystery(tree.right)
```

Circle the choice that correctly completes the statement below.

The function `mystery` returns:

(A) the width of the tree

(B) the number of elements that are `NonEmpty`

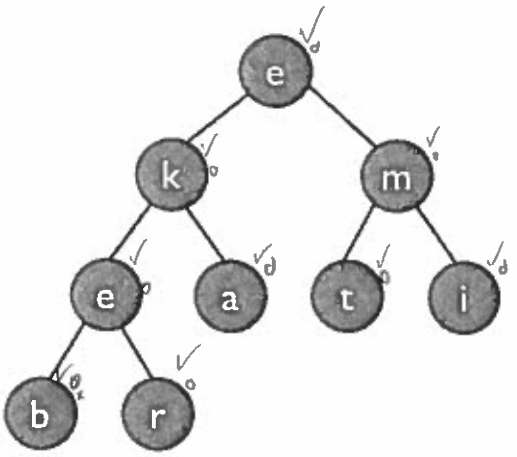
(C) the number of elements that are internal nodes

(D) the height of the tree

(E) the number of elements that are leaf nodes

(F) the number of elements in the tree

19. (3 points)



✓ pre ekebramti

o in order berkaetmi

x post breaktime

For each traversal, choose from this list of choices:

(A) timebreak
(B) kemeatibr
(C) kembearti
(D) ekmeatibr
(E) ekebramti pre
(F) breaktime post
(G) berkatmie
(H) berkaetmi in
(I) beakertim
(J) beakermit

3

Write the letter to the left of the traversal that is the correct match:

- The pre-order traversal of the above tree is: E
- The in-order traversal of the above tree is: H
- The post-order traversal of the above tree is: F

20. (2 points) The following code defines a function `minCoins`. It takes a single argument, a natural number representing an amount of money, and returns a natural number indicating the minimal number of coins that add up to the amount.

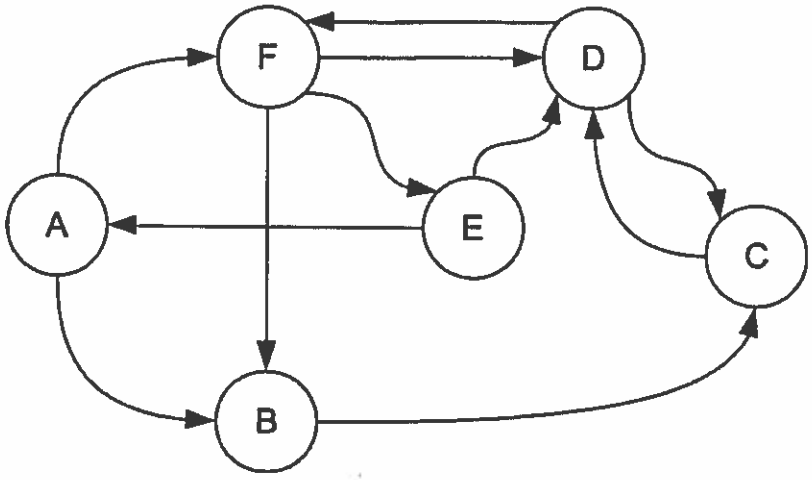
```
def minCoins(a):
    if a == 0:
        return 0
    elif 0 < a and a < 5:
        return 1 + minCoins(a - 1)
    elif 4 < a and a < 10:
        return 1 + minCoins(a - 5)
    elif 9 < a and a < 25:
        return 1 + minCoins(a - 10)
    else:
        return 1 + minCoins(a - 25)
```

- Show the substitution trace for `minCoins(65)`.

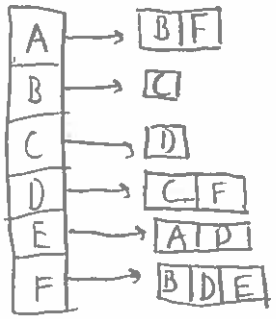
2

minCoins(65)
(1 + minCoins(40))
(1 + (1 + minCoins(15)))
(1 + (1 + (1 + minCoins(5))))
(1 + (1 + (1 + (1 + minCoins(0)))))
(1 + (1 + (1 + (1 + 0))))
(1 + (1 + (1 + 1)))
(1 + (1 + 2))
(1 + 3)
4

21. (2 points) Consider the following directed graph.



Give the adjacency-list representation of this graph. Put your answers in alphabetical order of nodes.



2

22. (2 points) Consider the following pseudocode that is supposed to visit each of the nodes in a graph once, if they are reachable from a starting node, and print their names.

```
function visit_node_helper( node )
    print( node.name )
    for neighbor_node in node.neighbors
        visit_node_helper( neighbor_node )

function visit_node( start )
    visit_node_helper( start )
```

Circle all of the following which provide an explanation of what might be wrong with the code or how to fix it.

(a) Only the original nodes get printed, not their neighbors.

(b) You need to keep a set of nodes that were already visited so you can skip them if you see them again.

(c) Unless you establish predecessor links on all the nodes, you'll never get back to where you started.

(d) If there is a cycle in the graph that is reachable from `start`, the code goes into an infinite loop and repeatedly prints the names of the nodes in the cycle.

23. (2 points) Consider the following pseudocode for an algorithm that acts on a weighted graph. Each edge has a first and second node and nodes have outgoing edges.

```
function doIt( start, visited, answer )
    for e in start.outgoingEdges
        answer = answer + e.weight
        n = e.secondNode
        if n is not in visited
            add n to visited
            answer = doIt( n, visited, answer )
    return answer


function doItSetup( start )
    return doIt( start, { start }, 0 )
```

Select the correct description of what the algorithm computes.

(a) the shortest path from start to any other node in the graph

(b) the concatenation of all the edges in the graph

(c) the sum of all edge weights in the graph

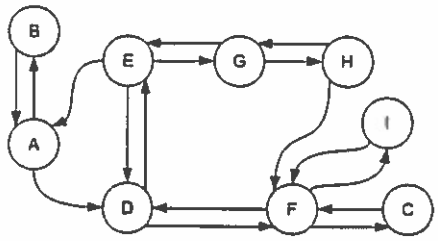
(d) the number of edges in the graph

(e) the longest path from start to any other node in the graph

(f) the sum of the lengths of all paths to all nodes from the start node

24. (4 points) Consider the following unweighted directed graph.

AB DEGHFC I



- (2 points) Perform a *recursive* depth-first search, from node A, to visit all nodes of the graph. Neighbors of a node should be processed in alphabetical order. The order in which nodes are added to the `visited` set is one (or more) of the following sequences of node names. *Circle exactly one correct sequence.*

4

A B D E F C I G H

A B D E F G C H I

A B D E F G C I H

A B D E G F H C I

– A B D E G H F C I

A B D F C I H E G

A B D F H I C E G

A I B D C E F H G

- (2 points) Perform a breadth-first search, from node A, to visit all nodes of the graph. Neighbors of a node should be processed in alphabetical order. The order in which nodes are added to the `predecessors` dictionary is one (or more) of the following sequences of node names. *Circle exactly one correct sequence.*

abdefgcih

– A B D E F C I G H

A B D E F G C H I

– A B D E F G C I H

A B D E G F H C I

A B D E G H F C I

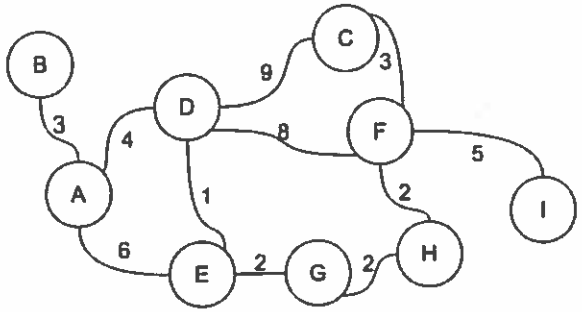
A B D F C I H E G

A B D F H I C E G

A I B D C E F H G

5

25. (5 points) Consider the following weighted, undirected graph:



(a) Apply Dijkstra's Algorithm to determine the minimum cost paths from vertex **E** to vertex **F**. Complete the following table, showing the finalized vertex and the (updated) distance and predecessor for each vertex at each iteration of the algorithm. The first row of the table has been completed. Stop as soon as you have finalized vertex **F**.

Finalized vertex	A	B	C	D ✓	E ✓	F	G ✓	H ✓	I
	∞, None	∞, None	∞, None	∞, None	0, None	∞, None	∞, None	∞, None	∞, None
E	6, E			1, E	✓		2, E		
D	5, D		10, D			9, D			
G								4, G	
H						8, H			
A		8, A							
F									

(b) The path from vertex **E** to vertex **F** is EGHF.

26. (2 points) The code below supports sharing a resource in a producer-consumer class of problems. Consider the following code snippet which is called from within a Thread's run method:

```
private boolean available = false;
// fields of shared resources/objects ...

public synchronized Object get() {
    while ( ! available ) {
       try {
          wait();
       } catch (InterruptedException e) { }
    }
    Object obj = ...  // fetch the shared resource here
    available = false;
    return obj;
}

public synchronized void put( Object obj ) {
    // add to the shared resource here
    available = true;
}
```

2

What is the synchronization problem with the above code?

The consumer thread is never woken up to check availability. Resources can be overwritten/lost before the consumer recieves them.

27. (4 points) Match the design pattern name with the definition or the situation that best describes one of its uses.

E Decorator

A Factory

B Iterator

C Observer

4

(A) Constructor calls by definition mention the name of the class to be instantiated. This pattern is useful when you don't want/need the client to know the specific concrete class being used.

~~B~~) Acting on each of the individual elements in a collection without having to know precisely how to access them.

~~C~~) When there is one object that several other objects need to be consistent with, that object notifies them so that they can update appropriately.

D) If you have several different algorithms for a particular problem (such as players of a game, or search algorithms), you can construct and use an interface so that one of these algorithms can be easily selected between at run time.

~~E~~) If you have a class that implements a basic interface, and you want to add additional functionality to that class, one approach is to design a new class that uses the original one and implements the same interface as the original one.

28. (1 point) Give one example of an exception class that is *checked*.

IOException

2

29. (1 point) Give one example of an exception class that is *unchecked*.

Null Pointer Exception

30. (2 points) Study the following code, which compiles and runs.

```
public class TProg3 extends Thread {
    public static final int NUM_THREADS = 5;
    public static int counter = 0;
    private int num;

    public TProg3( int num ) {
        this.num = num;
    }

    public void run() {
            while ( this.num != counter ) {
                yield();
            }
            System.out.println( num );
            counter = ( counter + 2 ) % NUM_THREADS;
    }

    public static void main( String args [ ] ) {
            for ( int i = 0 ; i < NUM_THREADS; i++ ) {
                Thread t = new TProg3( i );
                t.start();
            }
    }
}
```

0 1 2 3 4

- What is the output produced?

0
2
4 OK
1
3

31. (5 points) Match the concept with its definition. Choose the best description.

B inheritance

C abstract class

E polymorphism

A encapsulation

F interface

5

A) Objects inside are visible to each other but not from the outside unless explicitly made visible.

B) Allows refinements and additions to an existing set of defined states and behaviors.

C) May contain state and behavior but requires further implementation to be complete.

D) Assigns responsibility for an operation to another object.

E) A reference is able to refer to instances of more than one type as long as that instance's type conforms to the type of the reference. When a method call is seen in source code, you may not know what method will actually execute until the program runs and the actual class of the target object is known.

F) Specifies only what operations its implementations will support. You can implement more than one of these.

32. (2 points) Consider the following code snippet:

```
TreeMap<String,Integer> grades = new TreeMap<String,Integer>();
grades.put("Alice",96);
grades.put("Dilbert",92);
grades.put("Wally",71);
grades.put("Asok",88);
grades.put("Dilbert",94);
for (String name : grades.keySet()) {
    System.out.println(name + " got " + grades.get( name ) );
}
```

- Assuming the code is part of a program that compiles, what will it output?

Alice got 96
Asok got 88
Dilbert got 94
Wally got 71

2

33. (3 points) Show the output of the following program that uses exceptions.

```
public class ExDemo {
    public static void foo() {
        try {
            System.out.print("A ");
            int arr[] = new int[5];
            arr[5] = 10;
            System.out.print("B ");
        } catch (NullPointerException npe) {
            System.out.print("C ");
        } catch (IndexOutOfBoundsException iobe) {
            System.out.print("D ");
        }
        System.out.print("E ");
    }

    public static void bar() {
        try {
            System.out.print("F ");
            int x = 5 / 0;
            System.out.print("G ");
        } catch (ClassCastException cce) {
            System.out.print("H ");
        } finally {
            System.out.print("I ");
        }
        System.out.print("J ");
    }

    public static void main(String[] args) {
        try {
            System.out.print("K ");
            foo();
            bar();
            System.out.print("M ");
        } catch (ArithmeticException ae) {
            System.out.print("N ");
        } catch (Exception e) {
            System.out.print("O ");
        }
        System.out.println("P");
    }
}
```

K A D E I F N P

2½

34. (7 points)

The *knapsack problem* is a problem in combinatorial optimization: Given a set of unique items, each with a weight, determine which items to include in a collection so that the total weight is exactly equal to a given limit. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it to capacity.

Consider an instance of this problem that uses a knapsack that can hold a maximum weight of `10kg`, and the following set of items by `name, weight`:

(a) `Candlestick, 11kg`

(b) `Knife, 3kg`

(c) `Lead Pipe, 9kg`

(d) `Revolver, 5kg`

(e) `Rope, 2kg`

(f) `Wrench, 6kg`

Consider using a backtracking approach to solve the knapsack problem. Assume you begin with an empty knapsack.

(a) (2 points) List the set of items in the knapsack that represents a solution:

Knife, 3kg
Revolver, 5kg
Rope, 2kg

2

(b) In each of the following, circle exactly one answer.

- (1 point) A good description of each configuration used by the backtracking solver is
  - i. the current weight of items in the knapsack.
  - ii. a set of items that are in the knapsack, and a set of remaining items that are not in the knapsack.
  - iii. the number of items in the knapsack.
  - iv. a collection of sets of all possible combinations of items.

- (1 point) To generate the next configurations from the current configuration
  - i. for each item that has not been put in the knapsack, create new configurations where each each unplaced item is added individually to the items that are already in the knapsack.
  - ii. choose an item that is not in the knapsack and create on configuration for it where it is in the knapsack.
  - iii. choose an item that has not been put in the knapsack and create a new configuration for it.
  - iv. choose an item that is not in the knapsack and create two configurations for it, one where the item is in the knapsack and one where it isn't in the knapsack.

5

- (1 point) A goal configuration is a configuration where
  - i. all items have been placed in the knapsack.
  - ii. all items less than the capacity of the knapsack have been placed in the knapsack.
  - iii. the knapsack contains at least one item.
  - iv. the items in the knapsack sum up exactly to the capacity of the knapsack.

- (1 point) A good choice for a pruning strategy is to treat as invalid any configuration where
  - i. all items are in the knapsack.
  - ii. the first item in the set is not in the knapsack.
  - iii. the sum of the items in the knapsack exceeds the knapsack's capacity.
  - iv. the sum of the items in the knapsack is less than the knapsack's capacity.
- (1 point) In the general backtracking algorithm, the `solve` function invokes itself recursively in a loop. How do the results of those recursive invocations in the loop affect the result of the current invocation of `solve`?
  - i. The result is a list of answers from all of the recursive calls.
  - ii. The result is failure, if any recursive call returns failure.
  - iii. The result is failure, if all recursive calls return failure.
  - iv. The result is the non-failure answer from the first recursive call that does not return failure (if there is one).
  - v. (a) and (b)
  - vi. (c) and (d)

35. (6 points) Given this code:

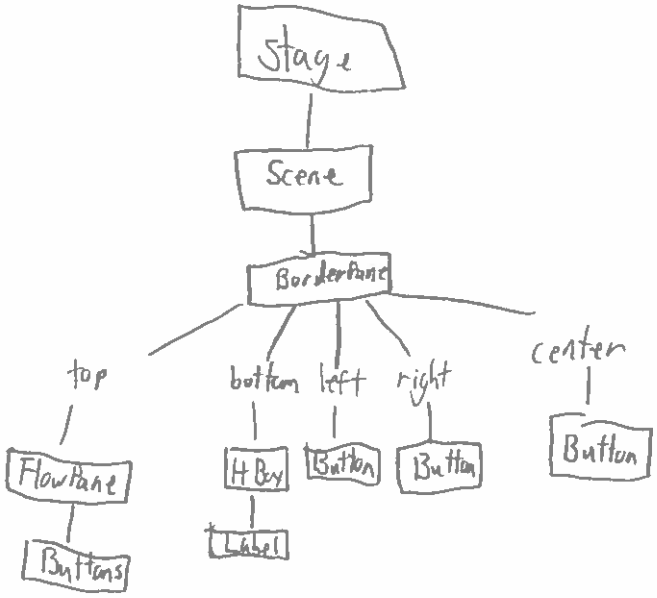
```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;

public class IsItGUI extends Application {

    public void start( Stage stage ) {

        BorderPane pane = new BorderPane();

        Pane pot = new FlowPane();
        pot.getChildren().addAll(new Button("<"), new Button(">") );
        pane.setTop(pot);

        pot = new HBox();
        pot.getChildren().add(new Label("...|...\\.../...|..."));
        pane.setBottom(pot);

        pane.setLeft(new Button("<<"));
        pane.setRight(new Button(">>"));

        Button theButton = new Button( "One" );
        theButton.setOnAction( null /* replace with handler */ );
        pane.setCenter(theButton);

        stage.setScene( new Scene( pane ) );
        stage.sizeToScene();
        stage.show();
    }
}
```

(a) Write a handler for `theButton` that can be inserted in place of the `null` in the code. When activated, the handler should change the button's text; if the text is "One", it should change to "THE One". If the text is "THE One", it should change to "One".

```
new ActionEventHandler<ButtonPressEvent>((e)->
    {public void handle(ButtonPressEvent e)
    { if ( theButton.text.equals("One"))
        { theButton.setText("THEOne");}
      else { theButton.setText("One");}
    }
```

3

(b) Draw the hierarchy of JavaFX components for this application starting from the `Stage` component.



3

36. (3 points) Match the concepts below with the best definition from the list below. Place the letter of your choice in the lines next to the concepts.

- I Stage
- F C Event Object
- G Event Handler
- C F ActionEvent
- K Observer
- A Model-View-Controller

2

Choose the *best* answer from the definitions below.

Definitions:

(A) A pattern that enables an application to implement multiple different UIs using the same application data.

Node? (B) The superclass for UI elements such as a button, text field or label.

(C) Will be sent to an event handler when a button is pressed.

scene? (D) The area that Nodes are placed into.

iterator? (E) A design pattern that allows a client to access/traverse the elements of a collection without the collection having to expose the underlying implementation.

(F) Generated by an event source. For example a button when pressed.

(G) The object that will be called when a UI component receives a stimulus.

(H) A class that is nested within another class definition.

(I) The top level component that represents a window with a title.

(J) A class used to organize the size and position of components within a display.

(K) A design pattern where a subject maintains a list of dependents and notifies them of any state changes.

Use this page for contemplation...