COMP 2140:
DATA STRUCTURES AND ALGORITHMS

SUMMER 2020

Old Exam Questions

O-NOTATION

SEE OTHER SECTIONS

RECURSION

PREDICT THE OUTPUT

For each block of code below, **state the output** that you would see on your screen. If the code will not compile or a run-time error will occur, **identify the code that will cause the problem and explain why it will not work**. Assume that any import statements needed are present, and that any methods are enclosed in a public class, appropriately named, so that any unseen code will not be the cause of compilation or run-time errors.

```
public static int test (int i){
    if ( i >= 10 )
        return 50;
    else
        return i + test( i + 3 );
}
public static void main( String[] args ){
    System.out.println( test(2) );
}
```

PREDICT THE OUTPUT

For each block of code below, **state the output** that you would see on your screen. If the code will not compile or a run-time error will occur, **identify the code that will cause the problem and explain why it will not work**. Assume that any import statements needed are present, and that any methods are enclosed in a public class, appropriately named, so that any unseen code will not be the cause of compilation or run-time errors.

```
public static int bar (int i){
    if ( i <= 0 )
        return 0;
    else
        return i + bar( i - 1 );
}
public static void main( String[] args ){
    System.out.println( bar(3) );
}
```

PREDICT THE OUTPUT

For each block of code below, **state the output** that you would see on your screen. If the code will not compile or a run-time error will occur, **identify the code that will cause the problem and explain why it will not work**. Assume that any import statements needed are present, and that any methods are enclosed in a public class, appropriately named, so that any unseen code will not be the cause of compilation or run-time errors.

```
public static int test2 ( int j ){
    if ( j == 2 )
        return 2;
    else
        return j * test2 ( j - 3 );
}
public static void main( String[] args ){
    System.out.println( test2(9) );
}
```

WRITE A RECURSIVE METHOD…

Write a recursive function that will return the value f(n) for any integer value of n, where f(n) is defined as:

If n <= 0: f(n) = 0
If n = 1: f(n) = 1
If n > 1 : f(n) = 3 * f(n - 1) + 2

WRITE A RECURSIVE METHOD…

Write a recursive function that accepts a String of any length and returns a boolean value, indicating whether the given string is a palindrome. (Recall that a palindrome is a word that reads the same forwards and backwards, such as "kayak" or "noon")

SHORT ANSWER

What is tail recursion? Describe how to rewrite code to remove tail recursion. Why would the rewritten version be more efficient?

SORTING

SHORT ANSWER

(a) Write a BRIEF (no more than 4 lines) description of the **selection sort** algorithm.

(b) For the given list of numbers, perform a **selection sort**, to sort the numbers from smallest on the left to largest on the right. Rewrite the list after each pass.

6 5 4 8 9 7 1

SHORT ANSWER

(a) Write a BRIEF description of the shellsort algorithm.

(b) For the given list of numbers, perform a **shellsort**, to sort the numbers from smallest on the left to largest on the right. Use the interval sequence h = 2 * h to determine the gaps. For each pass, identify the value of h used, and then rewrite the list of numbers as it would appear after the pass.

21 5 23 34 24 27 29 26 6 31 8 37

SHORT ANSWER

Explain the similarity and difference between Shellsort and Insertion sort. Why is Shellsort an improvement over Insertion Sort?

SHORT ANSWER

For the given list of numbers, perform a radix sort (with base-10 arithmetic). At each step, write a sentence describing how you are reordering the data and then rewrite the list of numbers.

475 215 441 224 421 175 115 124 021 275

SHORT ANSWER

For the given list of numbers, perform a **selection sort**, to sort the numbers from smallest on the left to largest on the right. Throughout the sort, keep track of any variable necessary to perform the sort. For each comparison, make a statement of the form "29 < 92" to demonstrate which values are being compared. For each swap, make a statement of the form "swap 29 and 92", and then rewrite the list of numbers with the stated change.

8 6 5 9 10 2

SHORT ANSWER

Fill in the blanks: Use the array containing 83, 27, 60, 19, 42, 51 (in that order) as the input for each question below:

(a) The very first pivot chosen by the quick sort algorithm we described in class, using the median-of-three technique, is _____ and the order of the items in the array after the very first partition is _____ _____ _____ (in that order).

(b) The merge sort public driver method (Call 1) calls the recursive merge sort method (Call 2), which calls itself recursively (Call 3). When Call 3 returns to Call 2, the array contains _____ _____ (in that order).

(c) After the first iteration of the outer loop of radix sort, the array contains _____ _____ (in that order).

LINKED LISTS

PREDICT THE OUTPUT

For each block of code below, **state the output** that you would see on your screen. If the code will not compile or a run-time error will occur, **identify the code that will cause the problem and explain why it will not work**. Assume that any import statements needed are present, and that any methods are enclosed in a public class, appropriately named, so that any unseen code will not be the cause of compilation or run-time errors.

Assume a Node class exists, with an integer data field, a reference to the next Node, and a getValue() method that returns the integer data stored in the Node, as seen in the 2140 course notes.

```
class CircularList {
        private Node first;
        private int size;
        public CircularList() { first = null;  size = 0; }
        public void add(int value){
                if (first==null){
                        first = new Node(value, null);  first.setNext(first);
                } else {
                        Node newNode = new Node(value, first.getNext());
                        first.setNext(newNode);
                }
                size++;
        }
        public void print(){
                Node current = first;
                while (current != null){
                        System.out.println(current.getValue());
                        current = current.getNext();
}       }       }
public static void main(String[] args){
        CircularList theList = new CircularList();
        theList.add(5);  theList.add(6);  theList.add(7);  theList.add(8);
        theList.print();
}
```

PROGRAMMING QUESTIONS

Write **Java** code to complete the tasks described in the questions below. You may assume that any necessary import statements are already present.

Given a linked list comprised of Nodes defined as below, and a pointer to the first node in the linked list (**`top`**), write a method (**`modifyList`**) to modify the given list in the following way. If the **`data`** in a given node is a positive number, insert **`data-1`** copies of that node after the given node. If the **`data`** in a given node is zero or negative, remove that node from the list. For example, if the initial list is

```
1   4   0   -3   2   -1
```

then the modified list would be

```
1   4   4   4    4    2   2
```

```
class Node {
    public int data;
    public Node next;
    public Node(int d, Node n){  data = d;  next = n;  }
}

void modifyList(Node top){

    //fill in...
```

PROGRAMMING QUESTIONS

Write **Java** code to complete the tasks described in the questions below. You may assume that any necessary import statements are already present.

Given a linked list comprised of Nodes defined as below, and a pointer to the first node in the linked list (**top**), write a method (**splitList**) to divide the list into two lists. Each node will either remain in the original list, or be moved into a new list. The first node (i.e. the node pointed to by top) and then every second node should remain in the original list. The other nodes should be placed in a new list. Nodes should remain in the same order as they were in the original list. **splitList** will return a pointer to the first node in the new list. For example, if the initial list is

```
1   4   0   -3   2   -1   8
```

then the modified list would be

```
1   0   2   8
```

and the new list would be

```
4   -3   -1
```

```
class Node {
    public int data;
    public Node next;
    public Node(int d, Node n){  data = d;  next = n;  }
}

Node splitList(Node top){

    //fill in...
```

SHORT ANSWER

When coding linked list methods, there are several special cases that you must watch for. For each of **TWO** such special cases: state the special case, state when you need to test for it, and state how you handle it.

PROGRAMMING

Write **Java** code to complete the tasks described in the questions below. You may assume that any necessary import statements are already present.

Given the Node class below, **write a complete class implementing a doubly linked list**, where the data held in each node is an integer, and the list is **ordered** from smallest to largest. Your code should be a complete implementation of the class and include the list's data members, a **constructor** that creates an empty list, and the **public methods**:

i) `void insert(int toInsert)`, which inserts the given value so that the list remains ordered by id,

ii) `void delete(int toRemove)`, which finds and removes the node with the given id, and

iii) `boolean find(int toFind)`, which searches for the given id and returns true if found, false otherwise.

Use good coding practice and avoid duplication of code by writing **a private search method** to be used by all three required methods.

Note that the data members of Node are public and you can directly access `data`, `previous`, and `next`.

```
class Node{
      public int data;
      public Node previous; //the node earlier in the list
      public Node next; //the next node in the list
      public Node(int newData, Node newPrev, Node newNext){
            data = newData; previous = newPrev; next = newNext;
      }
}
```

PROGRAMMING

Consider the circular linked list class on the separate sheets. Write a `CircularLinkedList` method with the header `public void append( CircularLinkedList secondList )`. This method has two circular linked lists (the calling list and a parameter list) and appends the second to the first. The result should be that the calling list is modified to contain all elements of the original calling list, followed by all elements of the original second list — the calling list's first node is still the first node, and the second list's last node becomes the calling list's last node — and the second list ends up empty.

GIVEN CODE FOR PREVIOUS QUESTION

For the circular-linked-list append question:

```
public class CircularLinkedList {

  private class Node {
    public int item;
    public Node next;
    public Node( int i, Node n ) {
      item = i;
      next = n;
    }
  } // end class Node

  private Node last;

  public CircularLinkedList() {
    last = null;
  } // end CircularLinkedList constructor

  public boolean isEmpty() { return last == null; }

  /*** Other CircularLinkedList operations would be placed here ***/
  /* (You MUST write your answers on the exam itself, not here.) */

} // end class CircularLinkedList
```

STACKS & QUEUES

PREDICT THE OUTPUT

INSTRUCTIONS FOR QUESTIONS 1 - 5: For each block of code below, **state the output** that you would see on your screen. If the code will not compile or a run-time error will occur, **identify the code that will cause the problem and explain why it will not work**. Assume that any import statements needed are present, and that any methods are enclosed in a public class, appropriately named, so that any unseen code will not be the cause of compilation or run-time errors.

Assume that Stack and Queue classes exist, with pop/push and enqueue/dequeue methods.

```
public static void main(String[] args){
  Stack theStack = new Stack();
  Queue theQueue = new Queue();
  theStack.push(9); theStack.push(3); theStack.push(7);
  theQueue.enqueue(8); theQueue.enqueue(2); theQueue.enqueue(6);
  System.out.println("Test 1: "+ theStack.pop() +" "+theQueue.dequeue());
  System.out.println("Test 2: "+ theStack.pop() +" "+theQueue.dequeue());
}
```

Output/Error:

SKETCH

Assume that we have a `Stack` class that implements ADT Stack using an array named `stackArray` as described in class. Assume the constructor is passed the array size to use. Consider the execution of the following code (and take note of the comments!):

```
Stack myStack = new Stack( 7 ); // Statement 1
myStack.push( 14 );
myStack.push( 2 ); // Statement 3
int myItem = myStack.pop();
myItem = myStack.top(); // Statement 5
```

Draw the `Stack` variable `myStack` and the entire stack that it points at after Statement 1 has been executed, after Statement 3 has been executed and after Statement 5 has been executed (*three* drawings). Anything that is a pointer must be explicitly represented as an arrow or `null`, as appropriate (never both at the same time!). You do not have to draw the variable `myItem`.

SHORT ANSWER

Fill in the blanks:

(a) The worst case running time of an enter (a.k.a. enqueue) operation in a queue containing n items is _____ if the queue is implemented as a circular linked list with just a `last` pointer and is _____ if the queue is implemented as an array with `front` and `end` indexes as described in class.

(b) At the end of each line of following code, fill in the blank with the text representation that shows what the queue contains after the line is executed. (Use the ⟨ ⟨ notation described in class.)

```
Queue myQueue = new Queue();  ______________________________________

myQueue.enter( 14 );  ______________________________________

myQueue.enter( -3 );  ______________________________________

myQueue.enter( 42 );  ______________________________________

int val = myQueue.leave();  ______________________________________

int val = myQueue.front();  ______________________________________

if ( ! myQueue.isEmpty() )  ______________________________________

  System.out.println( myQueue.leave() );  ______________________________________
```

HASHING

SHORT ANSWER

Assume a **hash table** is implemented using an array. In the 17 element array below, insert the following list of numbers using the hash function **`key % 17`**. Use double hashing, with the secondary hash function **`5-(key % 5)`**, to resolve collisions.

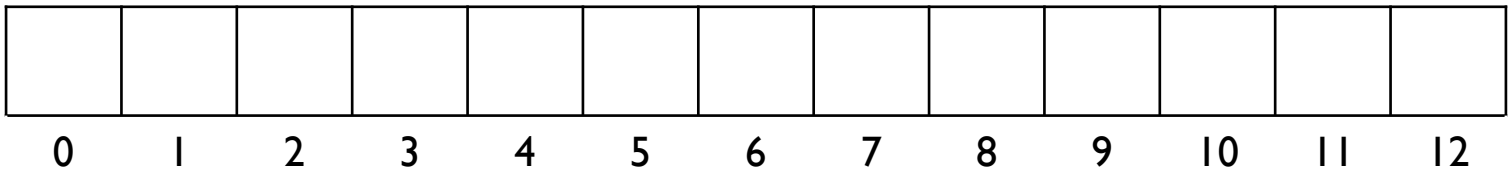
Numbers to insert in hash table: 8 36 2 19 1 35 18 23 6 27 5 30

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

SHORT ANSWER

- Assume a hash table is implemented using an array. In the 13 element array below, insert the following list of numbers using the hash function *h*(*key*) = *key* mod 13. Use double hashing, with the secondary hash function *h2*(*key*) = 5 – (*key* mod 5), to resolve collisions.

  Numbers to insert: 8, 39, 2, 15, 28, 21, 34, 26, 9, 12



SHORT ANSWER

With regard to hashing, what is probing? When is it used? Name one type of probing and describe it in sufficient detail that someone with a basic knowledge of hashing could understand.

PROGRAMMING (ALSO SEE NEXT SLIDE)

Consider the `Table` class on the separate sheets, which implements a `Table` as a hash table that stores integer items and uses linear probing to resolve collisions. We know that the efficiency of a hash table decreases as the load factor increases. To keep the load factor low, a hash array can be enlarged as the number of stored items grows. Write a `Table` method with header `public void rehash( int newTableSize )` to enlarge the array to the new table size, and rehash the table. Assume the new table size that the rehash function is passed is guaranteed to already be a prime number and larger than the old table size.

GIVEN CODE FOR PREVIOUS SLIDE

For the hashing question:

```
public class Table {

  private static final int EMPTY_SLOT = Integer.MIN_VALUE;
  private static final int INITIAL_TABLE_SIZE = 2003;

  private int[] hashArray;
  private int numItems;

  public Table( ) {
    hashArray = new int[ INITIAL_TABLE_SIZE ];
    for ( int i = 0; i < INITIAL_TABLE_SIZE; i++ ) {
      hashArray[i] = EMPTY_SLOT;
    } // end for
    numItems = 0;
  } // end Table constructor

  private int hash( int key ) {
    return key % hashArray.length;
  } // end hash

  public boolean isEmpty() {
    return numItems == 0;
  }

  public void insert( int newItem ) {
    int index;

    if ( numItems > hashArray.length / 2 ) {
      rehash( closePrime( 2*hashArray.length ) );
    } // end if

    index = hash( newItem );
    while ( hashArray[ index ] != EMPTY_SLOT ) {
      index = (index + 1) % hashArray.length;
    } // end while
    hashArray[index] = newItem;
    numItems++;
  } // end insert

  /*** Other Table operations would be placed here ***/
  /* (You MUST write your answers on the exam itself, not here.) */

} // end class Table
```

SHORT ANSWER

Fill in the blanks: Separate chaining is a technique used in hash tables to resolve _____ Using separate chaining, an item _____ (is or isn't?) stored directly in one of the positions in the hash array itself. Using separate chaining, the number of probes needed to find a particular item I _____ (can or cannot?) be affected by another item J that hashes to a different location in the array than item I.

BINARY TREES

PREDICT THE OUTPUT

```
class Node{
      public int data;
      public Node left;
      public Node right;
      public Node(int d){ data = d;  left = null;  right = null; }
}
class BinaryTree{
      private Node root;
      public BinaryTree(){  root = null;  }
      public void insert(int d){ ...Assume this method performs a valid insert into tree... }
      public boolean find(int key){ //return true if key is found in tree
            Node current = root;
            while (current.data != key){
                  if (current == null)
                        return false; //not found
                  else if (current.data > key)
                        current = current.left;
                  else
                        current = current.right;
            }
            return true; //found
      }
}
public static void main( String[] args ){
      BinaryTree b = new BinaryTree();
      b.insert(2);  b.insert(6);  b.insert(1);
      System.out.println( "Is 4 found in the tree?" + b.find(4) );
}
```

Output/Error:

SHORT ANSWER

What does it mean for a tree to be balanced? What advantage do balanced trees have over binary trees?

What does it mean for a tree to be complete? Give an example of a data structure that uses a complete tree.

SKETCH

Beginning with an empty **binary search tree**, draw the binary search tree after each insertion or deletion, in the order listed below. Use the result of one step as the starting point for the next step. On deletion, use the inorder successor to replace the deleted node, if necessary.

Step 1) Insert 30

Step 2) Insert 20

Step 3) Insert 10

Step 4) Insert 60

Step 5) Insert 45

Step 6) Insert 50

Step 7) Delete 30

Step 8) Insert 47

Step 9) Insert 65

Step 10) Delete 60

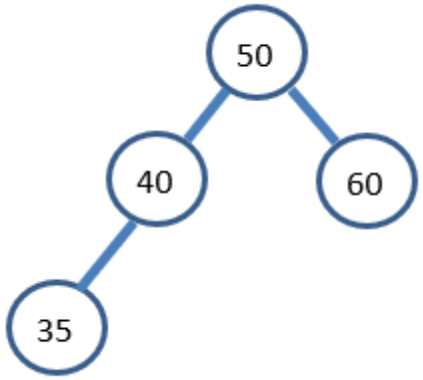
Step 11) Insert 48

Step 12) Delete 45

SKETCH

In each case below, beginning with the **binary search tree** shown on the left, on the right draw the binary search tree after the listed operation. On deletion, use the inorder successor to replace the deleted node, if necessary.

a) Given the binary search tree

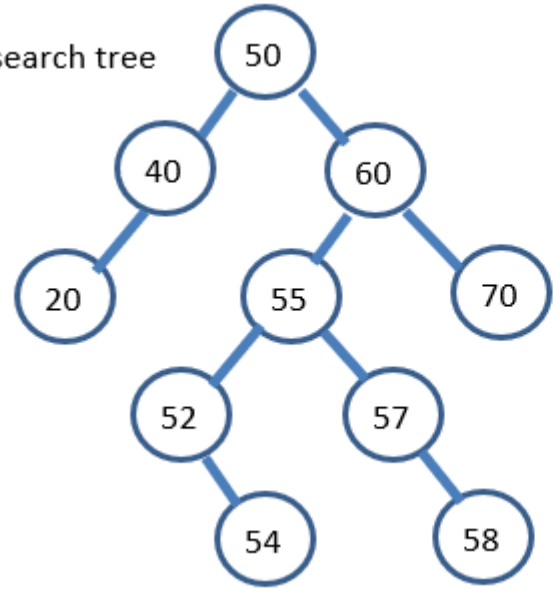


Insert 45:

SKETCH

In each case below, beginning with the **binary search tree** shown on the left, on the right draw the binary search tree after the listed operation. On deletion, use the inorder successor to replace the deleted node, if necessary.

b) Given the binary search tree

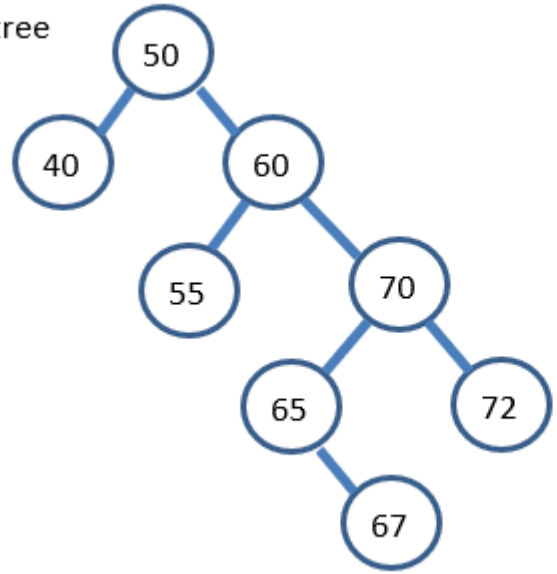


Delete 50:

SKETCH

In each case below, beginning with the **binary search tree** shown on the left, on the right draw the binary search tree after the listed operation. On deletion, use the inorder successor to replace the deleted node, if necessary.

c) Given the binary search tree

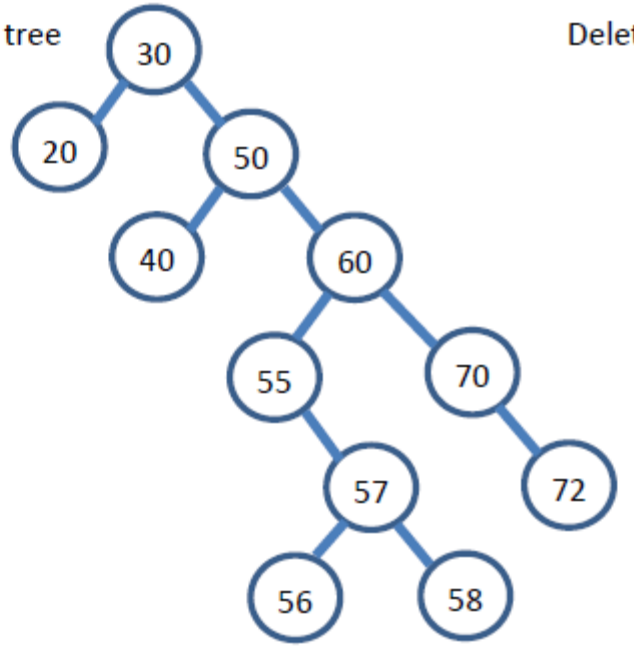


Delete 60:

SKETCH

In each case below, beginning with the **binary search tree** shown on the left, on the right draw the binary search tree after the listed operation. On deletion, use the inorder successor to replace the deleted node, if necessary.

c) Given the binary search tree



Delete 50:

PROGRAMMING

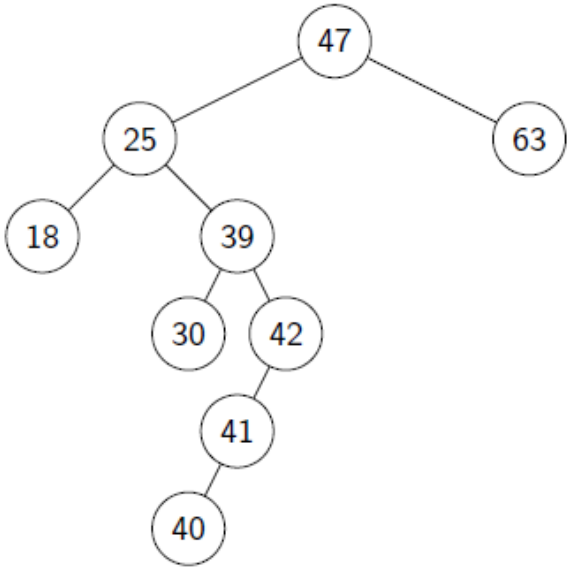
Write **Java** code to complete the tasks described in the questions below. You may assume that any necessary import statements are already present.

Write a method for a Binary Search Tree **to find the inorder successor to the node with a given key** (i.e. this method is part of the Binary Search Tree class). Assume that each Node in the tree has a public integer field (called "**data**") that is the key field, and that nodes in the tree are ordered by increasing key. Each Node also contains public references to two child Nodes ("**left**" and "**right**"). The method will **return the (integer) data field from the inorder successor**. The method signature is given as a starting point.

```
int  getInorderSuccessor( int key ) {
```

SHORT ANSWER

For each of the following sequences, state whether it is a preorder traversal or an inorder traversal or a postorder traversal of the following binary search tree, or if it is not any of the three traversals. (Assume that the traversal's job is simply to print out the values stored in the nodes.)



(a) 18 30 40 41 42 39 25 63 47 is _____

(b) 18 25 30 39 40 41 42 47 63 is _____

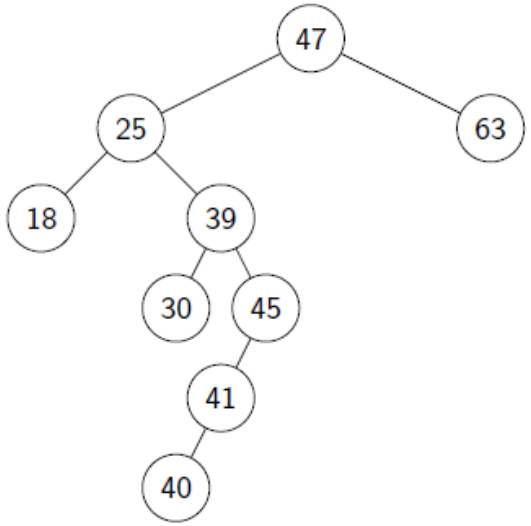
(c) 47 63 25 40 18 41 39 42 30 is _____

(d) 47 25 18 39 30 42 41 40 63 is _____

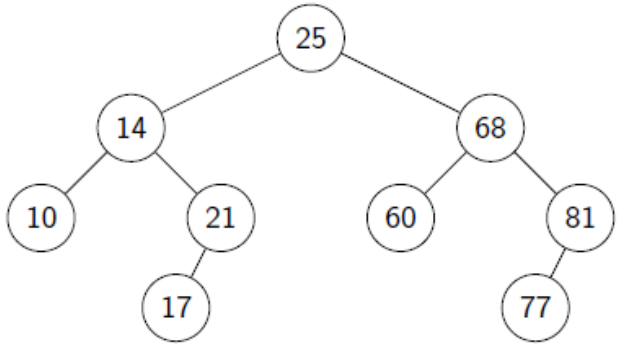
SKETCH

In each of the following questions, draw the binary search tree (BST) that results from inserting 42 into the given BST using the BST insertion algorithm described in class.

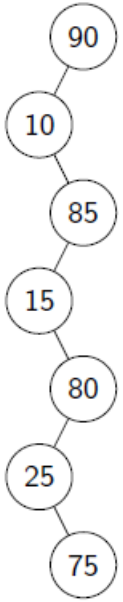
(a) Insert 42:



(b) Insert 42:



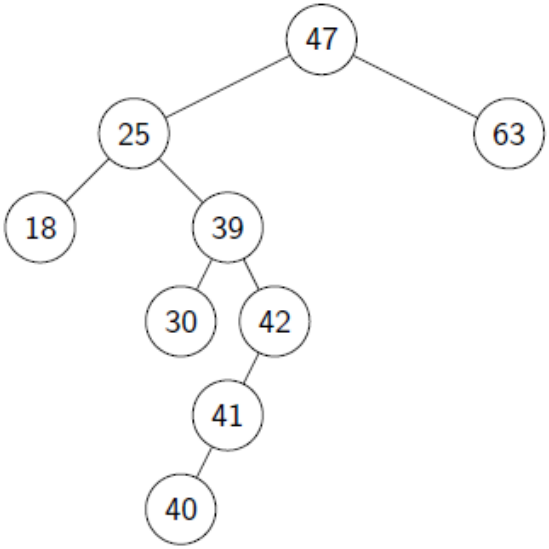
(c) Insert 42:



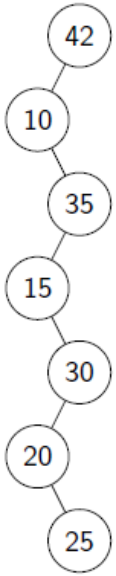
SKETCH

For each question, draw the binary search tree (BST) that results from deleting 42 from the given BST using the BST deletion algorithm described in class.

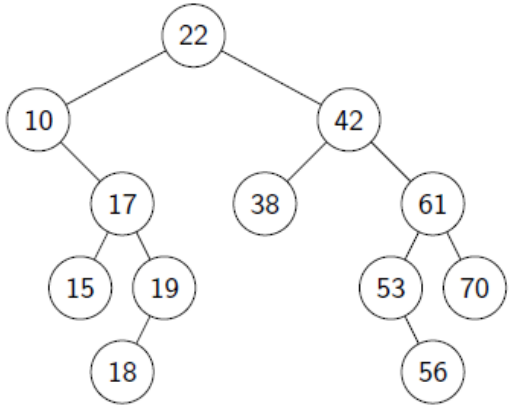
(a) Delete 42:



(b) Delete 42:



(c) Delete 42:



PROGRAMMING (ALSO SEE NEXT PAGE)

Consider the `BST` (binary search tree) class on the separate pages. Write a `BST` method `deleteOdd()` that deletes all odd values stored in the calling binary search tree. Hint: Use a post order traversal that returns the node that should replace the current node.

GIVEN CODE FOR PREVIOUS PAGE

```
public class BST {
  private class BSTNode {
    public int item;
    public BSTNode left;
    public BSTNode right;

    public BSTNode( int newValue ) {
      item = newValue;
      left = right = null;
    } // end BSTNode constructor

  } // end class BSTNode

  private BSTNode root;

  public BST() {
    root = null;
  } // end BST constructor

  public boolean isEmpty() { return root == null; }

  public void insert( int newValue ) { // No duplicates!
    BSTNode curr = root, prev = null, newLeaf;

    while ( curr != null && curr.item != newValue ) {
      prev = curr;
      if ( newValue < curr.item )
        curr = curr.left;
      else
        curr = curr.right;
    } // end while
    if ( curr == null ) {
      newLeaf = new BSTNode( newValue );
      if ( prev == null )
        root = newLeaf;
      else {
        if ( newValue < prev.item )
          prev.left = newLeaf;
        else
          prev.right = newLeaf;
      } // end if-else
    } // end if ( curr == null )
  } // end insert

  /*** Other BST operations would be placed here ***/
  /* (You MUST write your answers on the exam itself, not here.) */

} // end class BST
```

SKETCH

Draw a complete binary search tree that contains the values 1–11.

MULTIWAY TREES

SHORT ANSWER

What is a B-tree? Where is it used? What is the advantage over a binary tree?

SKETCH

Beginning with an empty **2-3 Tree**, draw the 2-3 Tree after each of the following insert operations, using the 2-3 tree resulting from one operation as the starting point for the next operation.

Step 1) Insert 30

Step 2) Insert 20

Step 3) Insert 10

Step 4) Insert 60

Step 5) Insert 45

Step 6) Insert 50

Step 7) Insert 15

Step 8) Insert 47

Step 9) Insert 65

Step 10) Insert 62

Step 11) Insert 49

Step 12) Insert 48

SKETCH

In each case below, beginning with the **2-3 Tree** shown on the left, on the right draw the 2-3 Tree after the listed operation.

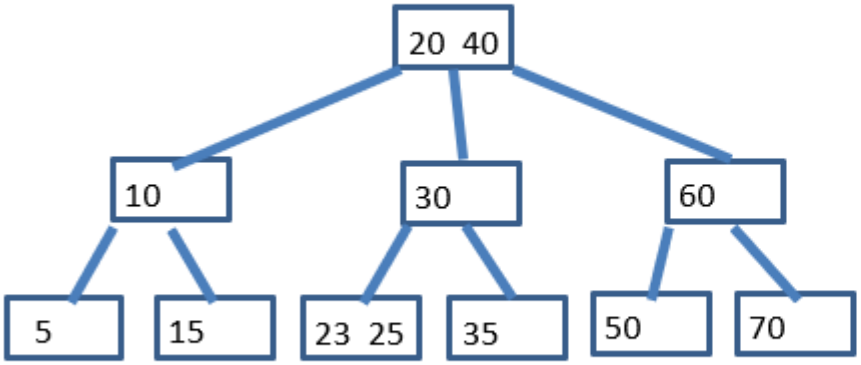
a) Given the 2-3 Tree

Insert 10:

SKETCH

In each case below, beginning with the **2-3 Tree** shown on the left, on the right draw the 2-3 Tree after the listed operation.

b) Given the 2-3 Tree

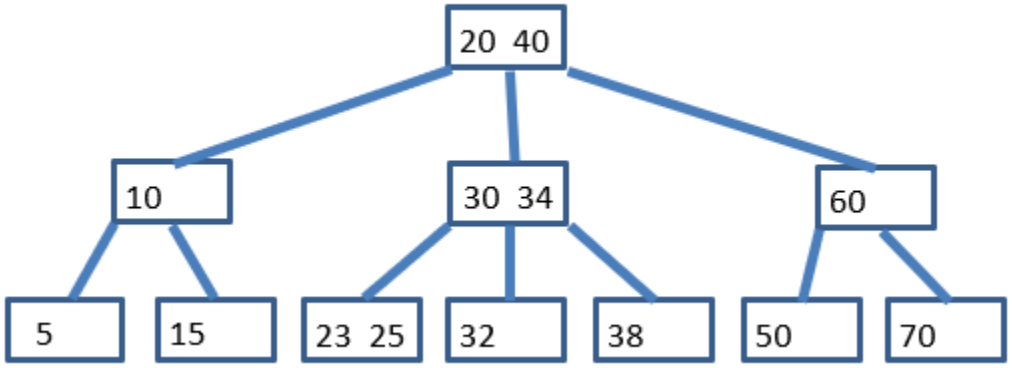


Insert 28:

SKETCH

In each case below, beginning with the **2-3 Tree** shown on the left, on the right draw the 2-3 Tree after the listed operation.

c) Given the 2-3 Tree

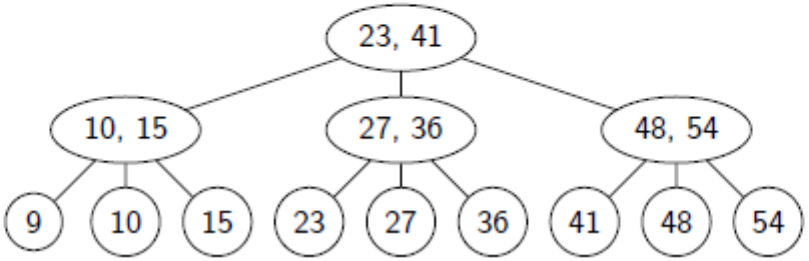


Insert 21:

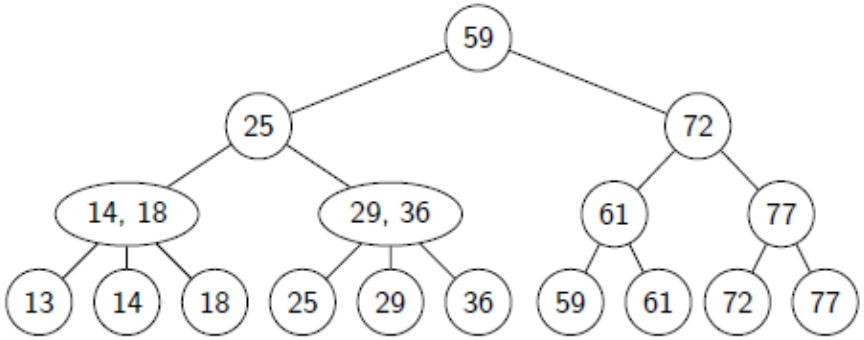
SKETCH

For each of the following questions, insert 42 into the specified leaf-based 2-3 tree using the algorithm described in class. Draw the leaf-based 2-3 tree that results from each insertion. (You can show intermediate steps if you want to, but make sure that you clearly mark which one is the resulting tree.)

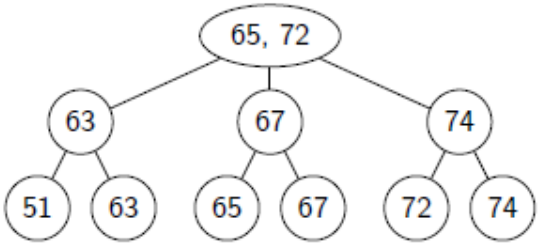
(a) Insert 42:



(b) Insert 42:



(c) Insert 42:



SHORT ANSWER

Fill in the blanks: For leaf-based 2-3 trees containing n data items, the worst-case running time of search is _____ and of insert is _____.

HEAPS & PRIORITY QUEUES

SHORT ANSWER

Often a heap is implemented as an array. Describe how this is accomplished. In particular, if we have the index **x** for a particular node in the heap, how do we locate the parent and the children of that node?

SKETCH

Beginning with an empty **descending-order heap**, draw the heap after each insertion or removal, in the order listed below. Use the result of one step as the starting point for the next step.

Step 1) Insert 30

Step 2) Insert 60

Step 3) Insert 10

Step 4) Insert 40

Step 5) Insert 50

Step 6) Insert 70

Step 7) Remove a node.

Step 8) Insert 20

Step 9) Insert 25

Step 10) Insert 45

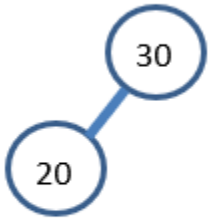
Step 11) Remove a node.

Step 12) Remove a node.

SKETCH

In each case below, beginning with the **descending-order heap** shown on the left, on the right draw the heap after the listed operation.

a) Given the heap

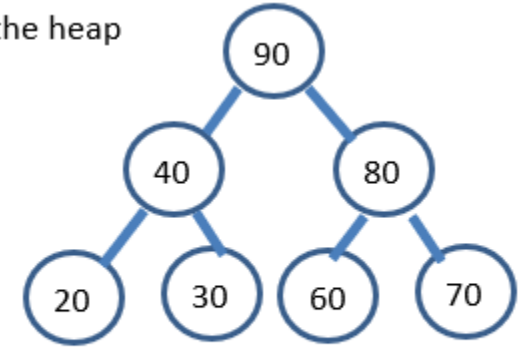


Insert 10:

SKETCH

In each case below, beginning with the **descending-order heap** shown on the left, on the right draw the heap after the listed operation.

b) Given the heap

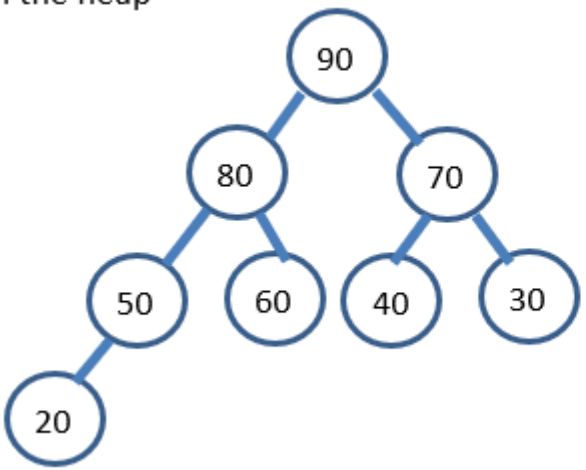


Remove a node:

SKETCH

In each case below, beginning with the **descending-order heap** shown on the left, on the right draw the heap after the listed operation.

c) Given the heap



Remove a node:

PROGRAMMING

Assume that a `Project` class exists, with instance variables that include `int id` (a unique project identifier), and `int priority`. Project priorities range from 1 to 10, where 1 is lowest priority and 10 is highest priority. You may assume that all fields in Project are public.

Also assume that a `PriorityQueue` class exists, where instance variables for the `PriorityQueue` class include `int currentSize` and the queue contents, stored in an **array of Projects** named **`qArray`**. The `PriorityQueue` class already has a constructor, enqueue, and dequeue methods that keep the queue contents ordered so that the highest priority project is at the front of the queue. If two projects have the same priority, the one added to the queue first will be closer to the front of the queue. Assume that the queue is stored in the array such that the front of the queue is always element 0 of the array.

Write a `promote` method for the PriorityQueue class that will accept the ID for a project, locate that project in the queue, and increase the priority by 1. After this promote method executes, the project should be the last of the projects with its new, higher priority, and have moved ahead of all projects with its old priority. (The exception is for projects that already have a priority of 10. No change should occur in that case.) Return a boolean value, indicating success at promoting the project with the given ID.

PROGRAMMING

Write a `static` method that is passed an array filled with integers. It should convert the unordered array into heap order. You may assume that copies of the `siftUp()` and `siftDown()` methods exist and have been made `static` also — you must pass them the array and an index and they will do a sift up or a sift down of that position in the array. Your method must modify the array in place — that is, you may not create any other arrays or other data structures to use in this method.

SKETCH

Perform the sequence of operations (below, on the right) in the given order on the max heap `myHeap` pictured below on the left. Draw the resulting heap (the array `heap` and `heapSize`) after all operations have completed. You can also show intermediate steps and the heap tree, if you wish, but make sure that you clearly indicate what is the final answer.

heapSize 7

heap

0	1	2	3	4	5	6	7	8	9
53	50	33	17	20	24	3			

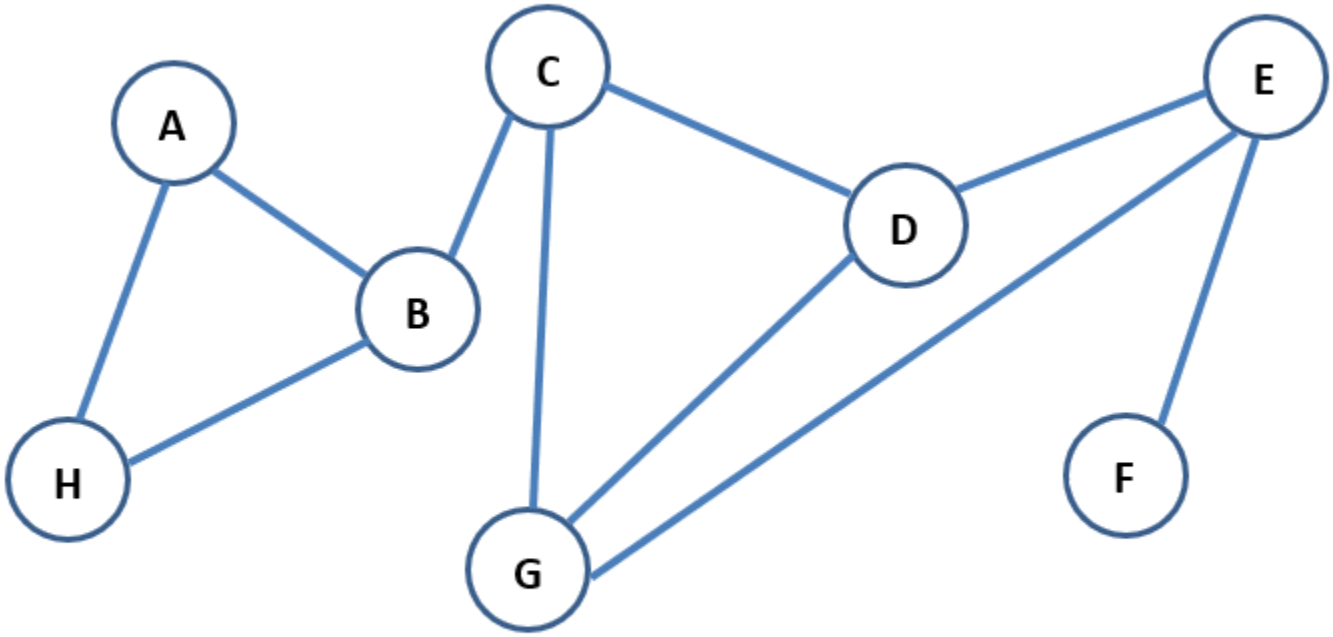
```
myHeap.deleteMax();
myHeap.deleteMax();
myHeap.insert( 42 );
myHeap.insert( 14 );
myHeap.deleteMax();
```

GRAPHS

From the given adjacency matrix, draw the associated graph.

	A	B	C	D	E
A	0	0	4	5	7
B	0	0	0	1	0
C	4	9	0	0	0
D	2	0	0	0	0
E	0	0	3	0	0

Perform a **depth-first traversal** of the graph below, **beginning at Node C**. List your result (i.e. the list of nodes in the order they are visited) below the graph. Assume the nodes are stored in alphabetical order.

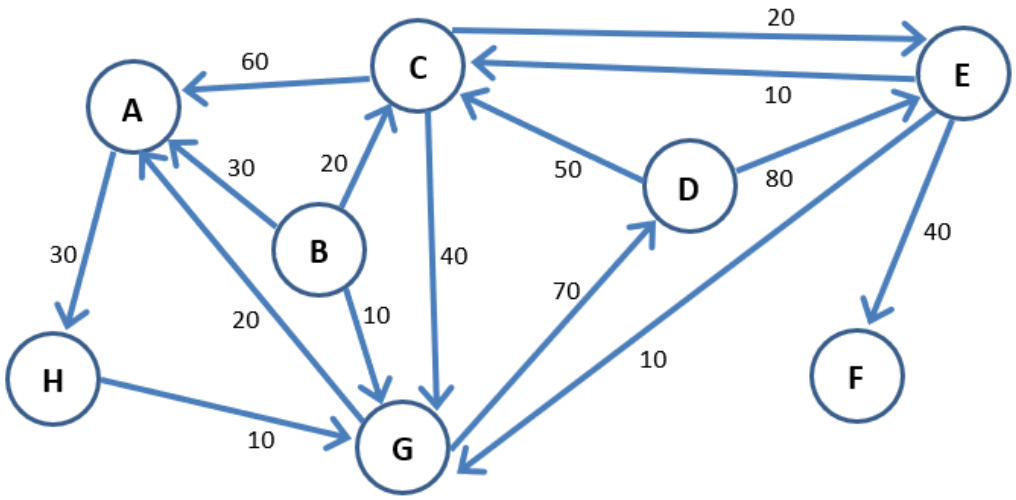


Result of depth-first traversal:

SHORT ANSWER

Describe Dijkstra's algorithm (in words, with pseudocode to assist your explanations if desired).

Use Dijkstra's algorithm on the graph below to find the shortest paths from vertex D to every other vertex. Fill in the table below the graph at each step during Dijkstra's algorithm. After completion, fill in the lists of nodes along the shortest path from D to each vertex, below the table.



From D →	A	B	C	E	F	G	H
Step 1: D							
Step 2:							
Step 3:							
Step 4:							
Step 5:							
Step 6:							
Step 7:							
Step 8:							

Shortest Paths:

D to A:

D to B:

D to C:

D to E:

D to F:

D to G:

D to H:

SKETCH

Consider the following adjacency matrix:

	0	1	2	3	4
0	0	0	4	5	7
1	0	0	9	1	0
2	4	9	0	0	3
3	5	1	0	0	0
4	7	0	3	0	0

(a) Draw the graph corresponding to the above adjacency matrix.

(b) Write the sequence of nodes that would result from a depth-first traversal of the graph starting at vertex 0, assuming that the traversal algorithm is using the adjacency matrix representation and that "visit the vertex" means "print out the name of the vertex".