University of Maryland College Park
Dept of Computer Science
CMSC132 Final Exam

Last Name (PRINT): _____ **First Name** (PRINT): _____

University Directory ID (e.g., umcpturtle): _____

Lab TA (Circle One):

0101/0102 Medhi	**0201Jeremy**	**0203/0204 Stefani**	**0303/0304 Raghav**	**0402 Solomon**	**0404 Victor**
0103/0104 Nisarg	**0202 William**	**0301/0302 Meena**	**0401 Avery**	**0403 Jihoon**	**Honors**

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Instructions

- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points. There are **eight** problems on the exam and one honor's problem.
- The exam is a 120 minutes exam.
- Please use a pencil to complete the exam. WRITE NEATLY.
- Your code must be efficient.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **Write your name and circle you section number before you answer the exam.**

Grader Use Only

#1	Algorithmic Complexity	18	
#2	Heaps	10	
#3	Software Engineering	20	
#4	Sorting/Hashing	14	
#5	Alg. Strategies/Design Patterns	10	
#6	Graphs	16	
#7	Linear Data Structures/Threads	72	
#8	Binary Trees	40	
	Honors	15	
Total	Total	200/215	

Problem #1 (Algorithmic Complexity)

1. (18 pts) For the following problems you need to provide the asymptotic complexity using Big O notation. In addition, you need to identify the critical section (circle it) and the time function (Time → below). Here is an example:

```
for (j = 1; j <= n; j++) {
    System.out.println(j);
}
System.out.println("Goodbye");
```

Time → n + 1
Big O → O(n)

a. (5 pts)

```
System.out.println("Start");
i = 1;
while (i <= n / 2) {
   for (k = 1; k <= n / 3; k += n) {
      System.out.println(i * k);
   }
   i++;
}
```

Time →
Big O →

b. (5 pts)

```
for (i = 10; i < n; i++) {
   for (k = 1; k <= n; k++) {
      System.out.println(i * k);
   }
}

for (i = 1; i <= n; i++) {
   for (m = n / 2; m >= 1; m--) {
      System.out.println(m * i);
   }
}
```

Time →
Big O →

c. (5 pts)

```
for (k = 1; k <= n; k *= 2) {
   System.out.println(k);
   for (m = n; m <= 2 * n; m++) {
      System.out.println(m);
   }
}
```

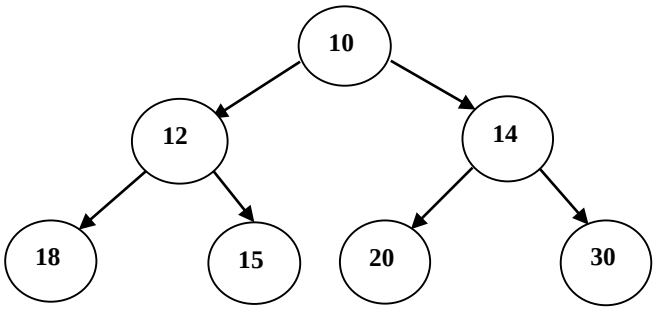
Time →
Big O →

2. (3 pts) List the following Big O expressions in order of asymptotic complexity (lowest complexity first) for size **n**.

$O(\mathbf{n}\log(\mathbf{n}))$ $O(\log(\mathbf{n}))$ $O(\mathbf{n!})$ $O(\mathbf{n^n})$

Problem #2 (Heaps)

Use the following heap to answer the questions that follow.



1. (1 pt) Specify the height of a heap using Big O notation.

2. (1 pt) Write the heap as an array.

3. (4 pts) Draw the heap (as a tree) that would result from inserting **11** in the above heap.

4. (4 pts) Draw the heap (as a tree) that would result by deleting **10 from the original heap** (not the heap from step 3.)

Problem #3 (Software Engineering)

1. (2 pts) Which of the following represent components of the software life cycle seen in class? Circle all that apply.

    a. Documentation and support
    b. Algorithms and data structures
    c. Backtracking development
    d. Divide and conquer
    e. None of the above

2. (2 pts) Which of the following are software process models seen in class? Circle all that apply.

    a. Formal methods
    b. Waterfall model
    c. Iterative development
    d. Recursive development
    e. Backtracking development
    f. None of the above

3. (2 pts) Which of the following are architectural styles discussed in class? Circle all that apply.

    a. Pipes and Filters
    b. Recursive
    c. Blackboard
    d. Greedy
    e. None of the above

4. (2 pts) Which of the following are examples of iterative software development seen in class? Circle all that apply.

    a. Waterfall
    b. Unified model
    c. Agile software development
    d. Divide and Conquer
    e. None of the above

5. (2 pts) Iterative software development emphasizes:

    a. Adaptability instead of predictability
    b. Predictability instead of adaptability
    c. Adaptability, predictability, and deployment
    d. None of the above

6. (2 pts) Which of the following apply to regression testing? Circle all that apply.

    a. Gives a developer much more freedom to change existing code
    b. Allows to check whether some functionality that used to work stops working
    c. Use mainly in Agile software development
    d. Not often used in the waterfall model
    e. None of the above

7. (2 pts) What is a flow path?

    a. All the execution sequences through a program
    b. Unique execution sequence through a program
    c. Sequence that allows program to finish
    d. Sequence that does not generate a cycle
    e. None of the above.

8. (6 pts) An electronic store sells two kinds of electronic devices: cell phones and car stereos. All electronic devices have a serial number, a cost, and a make. All electronic devices can be turned on and off. Cellphones have an antenna and a memory capacity, and car stereos allow us to tune into a specific radio station.

    Based on the above information, define an outline for the classes/interfaces you will need for the electronic devices available in the store. Keep in mind that you don't need to represent the store. You don't need to implement the classes/interfaces; just name the classes/interfaces and the methods/fields associated with the system. Provide an informal description of your system. For example, for a system that defines a cat, we expect something similar to the following:

    Class Cat - implements interface Animal
    Fields: name, breed
    Methods: eat, sleep

    Interface Animal …

Problem #4 (Sorting/Hashing)

1. (6 pts) Complete the following table.

Sorting Algorithm	Average Case Complexity	Worst Case Complexity
Quicksort		
Mergesort		
Selection Sort		

2. (2 pts) A sorting algorithm requires an array that is twice the size of the array we are trying to sort. Which sorting algorithm property or properties are associated with this algorithm? Circle all those that apply to this algorithm.

   a. In-place
   b. Stable
   c. Random
   d. None of the above

3. (2 pts) Which of the following sequences represent the result of applying the **quicksort partitioning** function to an array where the pivot value is 75? Circle all the valid ones.

   a. 5 2 **75** 80 100

   b. 2 5 **75** 100 80

   c. 80 **75** 2 5 100

   d. 100 5 **75** 80 2

4. (4 pts) Does the following class satisfy the The Java Hash Code contract? Yes or No answers without explanation will receive no credit.

```
public class Triangle {
   private int base, height, area;

   public Triangle(int base, int height) {
      area = base * height / 2;
   }

   public boolean equals(Object obj) {
      if (obj == this)
         return true;
      if (!(obj instanceof Triangle))
         return false;
      Triangle triangle = (Triangle) obj;

      return base == triangle.base && height == triangle.height;
   }

   public int hashCode() {
      return area;
   }
}
```

Problem #5 (Algorithm Strategies & Design Patterns)

1. (2 pts) Which algorithm strategy (in addition to recursion) seen in class is used by both quicksort and mergesort? Select only one.

   a. Backtracking
   b. Divide and Conquer
   c. Dynamic Programming
   d. Greedy
   e. Brute Force
   f. Branch and Bound
   g. None of the above

2. (2 pts) Which algorithm strategy (in addition to recursion) seen in class is used by Dijkstra's algorithm? Select only one.

   a. Backtracking
   b. Divide and Conquer
   c. Guess and Check
   d. Greedy
   e. Brute Force
   f. Branch and Bound
   g. None of the above

3. (2 pts) Which algorithm strategy (in addition to recursion) seen in class can be used to color a map using four colors? Select only one.

   a. Backtracking
   b. Divide and Conquer
   c. Dynamic Programming
   d. Greedy
   e. Brute Force
   f. Branch and Bound
   g. Heuristic

4. (2 pts) A class is implementing an interface that defines no methods. Which of the following design patterns is associated with this class and interface?

   a. State
   b. Singleton
   c. Marker
   d. Observer
   e. Iterator
   f. Decorator

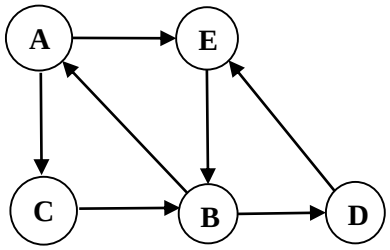
5. (2 pts) Which design pattern is associated with the following code?

```
BufferedReader bufferedReader = new BufferedReader(new FileReader("fileReader.txt"))
```

   a. State
   b. Singleton
   c. Marker
   d. Observer
   e. Iterator
   f. Decorator

Problem #6 (Graphs)

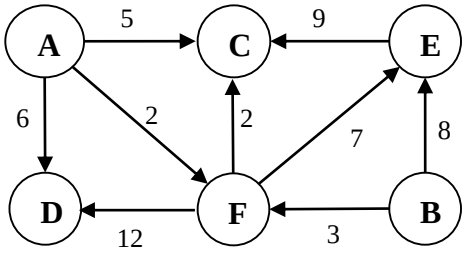
1. (4 pts) Graph traversals – For this problem you don't need to use the algorithms discussed in class. As long as your traversal is a valid BFS or DFS you will get full credit.



a. Give a possible order in which the nodes of this graph could be visited in performing a **breadth-first search (BFS)** starting at node **B**. Note: there may be more than one possible order that BFS could visit the nodes; you only have to provide one.

b. Give a possible order in which the nodes of this graph could be visited in performing a **depth-first search (DFS)** starting at node **B**. Note: there may be more than one possible order that DFS could visit the nodes; you only have to provide one.

2. (12 pts) Single source shortest paths



Apply Dijkstra's algorithm using **A** as the starting (source) node. Indicate the cost and predecessor for each node in the graph after processing two nodes (**A** and another node). Remember to update the appropriate table entries after processing a node (after it has been added to the set of processed nodes). An empty table entry implies an infinite cost or no predecessor. **Note: you will receive no credit if you simply fill in the entire table with the final costs and predecessors instead of showing the table at the first two steps.**

After processing the first node. Write the name of the node you are processing here: _____

Node	A	B	C	D	E	F
Cost						
Predecessor						

After processing the second node. Write the name of the node you are processing here: _____

Node	A	B	C	D	E	F
Cost						
Predecessor						

Problem #7 (Linear Data Structures/Threads)

Use the following classes to implement the methods below. Notice that in addition to the **head** reference the list has a **middleNode** reference that points to the middle node of the list. In addition, the number of elements in the list is represented by the **size** instance variable. You may not add any instance variables nor static variables to either class, and you may not add any methods to the Node class. **For any recursive function you may only add one auxiliary function**. You may not use the Java API LinkedList class.

```
public class LinkedList<T extends Comparable<T>> {
   private class Node {
      private T data;
      private Node next;

      private Node(T data) {
         this.data = data;
          next = null;
      }
   }

   private Node head, middleNode;
   private int size;

   public LinkedList() {
       head = middleNode = null;
       size = 0;
   }

   public Set<T> removeLargerOrEqual(boolean sorted, T lowerBound) { /* Implement */ }
   public LinkedList(Set<T> elements) { /* Implement */ }
   public void addWithThreads(Set<T> result) throws InterruptedException {/* Implement */ }
}
```

1. (23 pts) Provide a **RECURSIVE** implementation for the **removeLargerOrEqual** method. The method removes from the list elements that have a value larger or equal to **lowerBound**. In addition, a set with the elements removed will be returned. If the **sorted** parameter is true, a sorted set will be returned; otherwise the most efficient set will be returned. **You will lose most of the credit if you use any iteration statement (while, do while, for).** You do not have to worry about setting the middleNode.

2. (27 pts) Provide a **NON-RECURSIVE** implementation for the **LinkedList** constructor that takes a set as a parameter. For this problem:

   a. The set parameter is not sorted.
   b. Elements must be added so the list in **increasing** sorted order (e.g., "A" appears before "B").
   c. You may only perform a single iteration over the set.
   d. You may not use TreeMap, Collections.sort or any other sort method.
   e. The middle element is defined as the number of elements / 2 + 1;

3. (22 pts) The method **addWithThreads** collects (adds) all the elements of the list into the provided set, by using two threads. One thread will process all the elements before middleNode and the second all the elements after (including) middleNode. For this problem:

   a. You can assume the **middleNode** reference has been set correctly.
   b. You may not create new sets; if you do you will lose significant credit.
   c. You must use a lambda expressions to define the task each thread will perform; you will lose some credit if you don't use them.
   d. Feel free to add any methods/classes to the LinkedList class provided.

PAGE FOR YOUR ANSWERS

We are providing a driver (and expected output) that illustrates the functionality of the methods you are expected to implement. You can ignore it if you know what to implement. Notice we are using methods (e.g., fromMiddleOn() you don't need to implement).

```
Set<String> set = new HashSet<String>();
for (String str : new String[] { "C", "A", "D", "B" })
       set.add(str);
for (String str : new String[] { "I", "G", "F", "H", "E" })
       set.add(str);

LinkedList<String> list = new LinkedList<String>(set);
System.out.println("List: " + list);
System.out.println("Middle point on: " +
list.fromMiddleOn());

TreeSet<String> addedToSet = new TreeSet<String>();
list.addWithThreads(addedToSet);
System.out.println("Threads results: " + addedToSet);

set = list.removeLargerOrEqual(true, "D");
System.out.println("List after removeLarger: " + list);
System.out.println("Set returned (removeLarger): " + set);
```

```
List: Size: 9, Elements: A B C D E F G H I
Middle point on: Size: 9, Elements: E F G H I
Threads results: [A, B, C, D, E, F, G, H, I]
List after removeLarger: Size: 3, Elements: A B C
Set returned (removeLarger): [D, E, F, G, H, I]
```

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS

Problem #8 (Binary Trees)

Use the following classes to implement the methods below. You may not add any instance variables nor static variables to either class, and you may not add any methods to the Node class. You may not use the Java API maps/set classes. **For any recursive function you may only add one auxiliary function**. **You will lose significant credit if we ask for a recursive implementation and you use any iteration statement (while, do while, for) in your solution.**

```
public class BinarySearchTree <K extends Comparable<K>, V> {
   private class Node {
      private K key;
      private V data;
      private Node left, right;

      public Node(K key, V data) {
         this.key = key;
         this.data = data;
      }
   }

   private Node root;
   public ArrayList<K> getIncreasingKeyList() { /* Implement */ }
   public void processNodesOneChild(Task<K,V> task) { /* Implement */ }
}

public interface Task<K,V> {
   public void processing(K key, V data);
}

public class PrintTask<K, V> implements Task<K,V> {
   public void processing(K key, V value) {
      System.out.println(key + ", " + value);
   }
}
```

1. (10 pts) Provide a **RECURSIVE** implementation for the **getIncreasingKeyList** method. The method returns an ArrayList with the keys of the tree in increasing order.

2. (30 pts) Provide a **RECURSIVE** implementation for the **processNodesOneChild** method. The method will call the **processing** method (associated with the **Task** interface) on nodes of the tree that have only one child. For example, assuming **bst** is a BinarySearchTree object, calling

   bst.processNodesOneChild(new PrintTask<Integer, String>());

   will call the processing method associated with the PrintTask class. For this problem you can process nodes in any order.

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS

Honors

Students in the Honor's section must answer this question and only students in the Honor's section will receive credit for answering it.

Provide a recursive implementation for the **getNodesLevel** method that belongs to the previous BinarySearchTree class. The method returns a set with the keys of nodes that are at a specific level. For this problem:

a. The root is at level 1.
b. You can return any kind of set.
c. Your solution must be recursive.
d. Return an empty set if the tree is empty.

```
public Set<K> getNodesLevel(int targetLevel)
```