Your exam shall be scanned. Please write DARKLY and NEATLY.
Please PRINT your name and UTEID clearly in the boxes provided.

Your Name:

Your UTEID:

Instructions:

1. Write your exam number at the bottom of pages 3, 5, 7, 9, and 11 in the spaces provided.
2. There are **4** questions on this test. 100 points available.
3. You have 2 hours to complete the test.
4. Place your final answers on this test. **NO SCRATCH PAPER SHALL BE ACCEPTED FOR GRADING. You cannot add pages to the exam.**
5. **This exam is 100% closed**. You may not use **outside resources of any kind** (no electronic devices, computing devices, notes, other people, etc.) while taking the exam.
6. When answering coding questions, ensure you follow the restrictions of the question.
7. Do **not** write code to check the preconditions on the coding questions, 2 - 4.
8. On coding questions, you **may** implement your own helper methods.
9. On coding questions, make your solutions as efficient as possible given the restrictions of the question.
10. **Exam proctors shall not address any questions regarding the content of the exam. If you think a question is ambiguous or has an error, state your assumptions and answer based on those assumptions.**
11. When you complete the test show the proctor your UTID and give them the test. Please place used and unused scratch paper in the appropriate boxes at the front of the room. Please leave the room quietly

1. (2 points each, 50 points total) Short answer. Place your answer **in the box** next to or under the question. **Assume all necessary imports have been made.**

   a. If a question contains a syntax error or compile error, answer **compile error**.
   b. If a question would result in a runtime error or exception, answer **runtime error**.
   c. If a question results in an infinite loop, answer **infinite loop**.
   d. Recall, when asked for Big O your answer shall be the most restrictive correct Big O function. Closest without going under.
   e. Assume $\log_2(1{,}000) = 10$ and $\log_2(1{,}000{,}000) = 20$.

A. What is the order (Big O) of the following method? It uses the **`LinkedList314`** class from lecture. N = **`list.size()`**.

```
public static int a(LinkedList314<Integer> list) {
    int t = 0;
    for (int i = 1; i < list.size(); i *= 2) {
        t += list.get(i);
    }
    return t;
}
```

B. When the following code is run, what is the **sum of the values** in the nodes with a depth of 2? The **`BST314`** class use the simple, naïve insertion algorithm demonstrated in lecture.

```
BST314<Integer> tree1 = new BST314<>();
int[] data = {5, 2, 10, 5, 4, 12, -12, 6, 10, 9};
for (int x : data) {
    tree1.add(x);
}
```

C. When the following code is run, what is the height of the resulting tree? The **`BST314`** class use the simple, naïve insertion algorithm demonstrated in lecture.

```
BST314<Integer> tree2 = new BST314<>();
for (int i = 0; i < 1000; i++) {
    int val = i % 6;
    tree2.add(val);
}
```

D. The following method takes 10 seconds when **`n`** = 50,000. What is the expected time for the method to complete when **`n`** = 150,000? The **`BST314`** class **`iterativeAdd`** method uses the simple, naïve insertion algorithm and is implemented iteratively, not recursively.

```
public static BST314<Integer> d(int n) {
    BST314<Integer> res = new BST314<>();
    for (int i = 0; i < n; i++) {
        res.iterativeAdd(i);
    }
    return res;
}
```

E. The following method takes 10 seconds when **`n`** = 1,000,000. What is the expected time for the method to complete when **`n`** = 4,000,000? The method uses the **`java.util.TreeSet`** class.

```
public static TreeSet<Integer> e(int n) {
    TreeSet<Integer> res = new TreeSet<>();
    for (int i = 0; i < n; i++) {
        res.add(i);
    }
    return res;
}
```

F. After the following code is run, how many nodes are colored red in the resulting tree? The **RedBlackTree314** uses the Red-Black tree add algorithm as demonstrated in lecture.

```
RedBlackTree314<Integer> tree3 = new RedBlackTree314<>();
int[] data2 = {2, 5, 10, 5, 2, -4, 13};
for (int x : data2) {
    tree3.add(x);
}
```

G. On assignment 10, you implemented a program that created new encodings using the Huffman coding algorithm. Given **BITS_PER_WORD = 8**, as in the assignment, how many bits are required to encode the Huffman code tree in the **STF** format required by the assignment given the **input file contains 15 distinct values**? **Do not** include the 32 bits for the size of the tree that are written to the header before the bits that actually describe the tree, only the number of bits needed to represent the tree per the assignment 10 specification.

H. On assignment 10, you implemented a program that created new encodings using the Huffman coding algorithm. Assume an input file has only 5 distinct values with the following frequencies.

Value	Frequency
65	10
66	20
67	40
68	80
69	160

Using the algorithm specified on the assignment to build the Huffman code tree, what is the new binary encoding for the value 67?

I. Recall on assignment 8, you implemented the **AbstractSet** class, a skeletal implementation of the **ISet** interface. Recall, the goal was to implement as many of the methods from the **ISet** interface without an internal data structure and without explicit references to subclasses. Which of the following methods could **NOT** be implemented correctly in **AbstractSet** given those restrictions? **List the letters of all methods that could not be implemented. If none, answer NONE.**

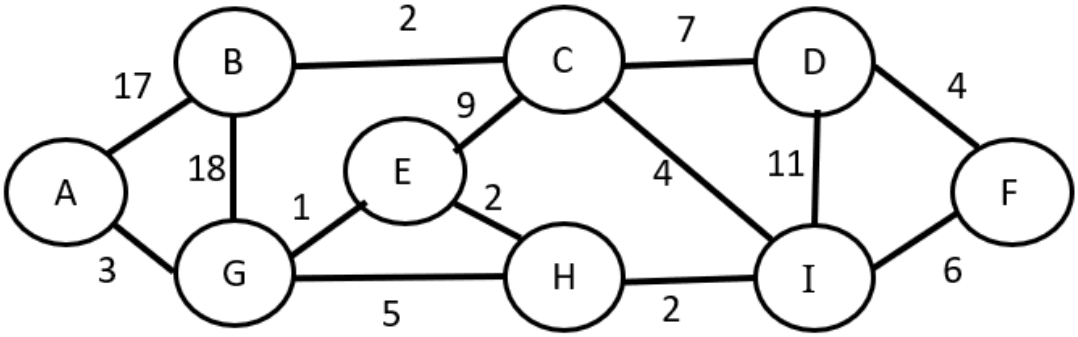
A. add B. addAll C. intersection D. iterator E. remove F. size

J. Recall the **Graph** and **Vertex** classes from assignment 11. The **Graph** class stores all **Vertex** objects in a variable of type **HashMap<String, Vertex>.** Given **V** vertices in the graph and each vertex having an average of **E** edges, what would the order (Big O) be of determining the maximum outdegree of **Vertex** objects in the **Graph**. Recall outdegree is the number of edges emanating out of a vertex in a directed graph. **Use the variables V and / or E in your answer, not N or M.**

EXAM # _____

K. Recall the **`Graph`** and **`Vertex`** classes from assignment 11. The **`Graph`** class stores all **`Vertex`** objects in a variable of type **`HashMap<String, Vertex>`**. Given **V** vertices in the graph and each vertex having an average of **E** edges, what would the order (Big O) be of determining the maximum indegree of **`Vertex`** objects in the **`Graph`**. Recall indegree is the number of edges leading to a vertex in a directed graph. **Use the variables V and / or E in your answer, not N or M.**

Graph for questions L, M, and N



L. What is the cost of the shortest (lowest cost) path from Vertex A to Vertex B?

M. Assume we use C as the starting vertex and perform Dijkstra's algorithm as presented in lecture to find the shortest (lowest cost) paths from Vertex C to all other vertices in the graph. What are the first four vertices visited (added to the **initially empty set** of vertices for which we have found the shortest (lowest cost) path) as the algorithm is completed? List the letters of the 4 vertices left to right in the order added. Do not include any spaces or commas in your answer.

N. What is the sum of the weights (costs) of the edges that form the minimum spanning tree of the graph above?

O. Recall the unsorted set we implemented in lecture. We used a hash table as the internal storage container. What is the worst-case order (Big O) of adding an element to the hash table we implemented? The hash table already contains N elements.

P. Consider a hash table implementation that uses probing to resolve collisions. (open addressing) Each element in the internal array contains at most one element. The hash table implements the **`Iterable`** interface. I claim using an iterator to access all of the elements in the hash table is O(N). Which of the following best describes what N represents? Pick the letter of the best answer.

A. The length of the internal array
B. The number of elements in the hash table from the client's perspective
C. The number of **`null`**s stored in the internal array'
D. The load limit of the hash table
E. The current load factor of the hash table

Q. If a hash table uses chaining (closed addressing) to resolve collisions, has a load limit of 2.0, a current load factor of 1.5, and currently contains N elements, what is the average case order (Big O) of adding an element to the hash table that is not already present?

R. Given a dictionary of the 150,000 most used words in English, we consider storing the words as a set of **`String`** objects, one **`String`** object for each word, OR store the words in a **`Trie`**. Let **S** be the sum of the lengths of the stored **`String`** objects and **T** be the number of node objects in the resulting **`Trie`**. Which one of the following statements is most likely correct?

A. **T** will be significantly less than **S**
B. **T** and **S** will be roughly the same.
C. **T** will be significantly more than **S**.

S. I claim searching a **`Trie`** to determine if a word that is represented by a **`String`** is present in the **`Trie`** is worst-case O(N). The nodes in the **`Trie`** use **`HashMap`**s to store references to child nodes. Which of the following best describes what N represents? Pick the letter of the best answer.

A. The number of words the **`Trie`** stores.
B. The number of nodes in the **`Trie`**.
C. The length of the word we are searching for.
D. The number of letters in the alphabet we are using.
C. The number of leaves in the **`Trie`**.

T. 11 integer values are inserted into an initially empty min-heap. How many leaves are in the resulting tree?

EXAM # _____

U. In a ternary heap (also known as a 3-heap) each node has at most 3 children. (left, middle, right) The ternary heap is complete in the sense that nodes are added left to right, top to bottom. We use an array to store the elements of the heap as we did in the lecture on heaps and the root element is at index 1. What is the correct formula to determine the index, **P**, of a parent node, given the index **N** of the child node? In other words, **P = ?**

A. `N / 3` B. `N / 2` C. `(N / 3) + 1` D. `(N + 1) / 3` E. `N % 3`

V. The following values are added in the order shown, left to right, to an initially empty min heap using the algorithm demonstrated in lecture. How many total swap operations are performed when adding the elements listed? Swapping 2 elements between a pair of nodes counts as 1 swap operation.

`5, 10, 8, 12, 18, 5, 20, 2, 8`

W. We are creating a new file format to store information about students at various colleges and universities. We want to use the minimum bits necessary when encoding data. We have 500 distinct colleges and universities to encode. What is the minimum number of bits necessary to encode the 500 colleges and universities?

X. In the second lecture on dynamic programming we went over recursive backtracking and dynamic programming solutions to the 0-1 knapsack problem. Given **N** items and a knapsack with **C** capacity what was the time order (Big O) of the two approaches? Items' weight must be integers. Pick the letter of the best answer.

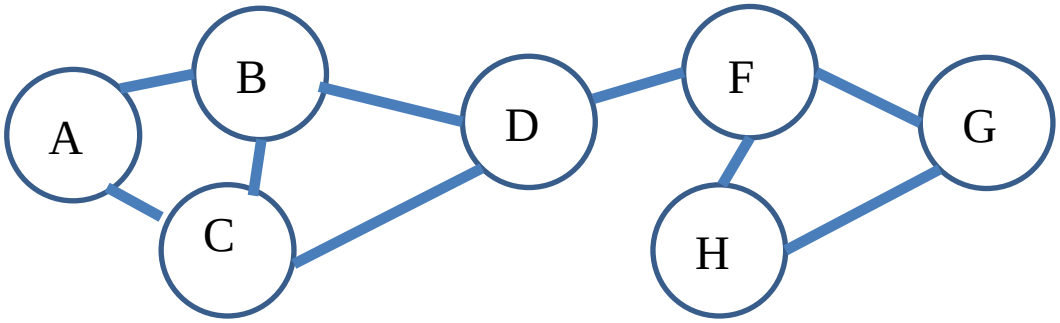
	Recursive Backtracking Solution	Dynamic Programming Solution
A.	$O(N^2)$	$O(N^2)$
B.	$O(2^N)$	$O(N^2)$
C.	$O(N^2C^2)$	$O(NC)$
D.	$O(2^N)$	$O(NC)$
E.	$O(C^N)$	$O(N^2C^2)$

Y. What is output by the following code?

```
int r =
    IntStream.range(4, 9)
    .map(x -> x * 2 - 3)
    .filter(y -> y < 9)
    .sum();
System.out.print(r);
```

Z. Extra Credit. When UT Austin was connected to the ARPANET circa 1978, what was the number assigned to the IMP (Interface Message Processor, the routers of the ARPANET) at UT Austin?

2. **Graphs** (17 points) An undirected graph is connected if every pair of vertices in the graph are connected. A pair of vertices are connected if there is a path from the first vertex to the second vertex. The graph below is an undirected, unweighted graph that is connected.



In a connected graph an edge is a ***bridge*** if removal of that edge causes the graph to no longer be connected. In the graph shown above, the edge connecting D and F is a bridge. If it is removed, the graph is no longer connected. The graph now consists of two components or subgraphs. The edge connecting D and C is not a bridge because even if it is removed all pairs of vertices are still connected. **Complete a method that determines if a given edge, specified by the two adjacent vertices is a bridge edge or not.**

The **`Graph`**, **`Vertex`**, and **`Edge`** classes for this question are essentially the same as from assignment 11. However, the **`Vertex`** class has an added method, **`removeEdge`**. The method removes and returns the edge with the given destination **`Vertex`** from the calling **`Vertex`** object's **`adjacency List`**.

```
public class Graph {
    private Map<String, Vertex> verts; // keys are names of vertices

    private void clearAll() // Sets all Vertex.scratch variables to 0.

    public boolean isBridge(String v1, String v2) // TO DO

    private static class Vertex {
        private String name;
        private List<Edge> adjacent;
        private int scratch;
        public Edge removeEdge(Vertex dest)
        // Do not use or add other instance variables or methods.
    }

    private static class Edge {
        private Vertex dest;
        // Do not use or add other instance variables or methods.
```

EXAM # _____

- The graph is unweighted and undirected. All edge costs are 1. If a vertex A has an edge in its adjacency list to vertex B, B also has an edge in its adjacency list to vertex A.
- The graph is connected.
- The two vertices specified by the two parameters to the **`isBridge`** method are adjacent.
- **Do not use recursion.** Implement a breadth first search. You may create and use a single instance of the **`Queue314`** class, a fair queue. You may use the following methods: **`enqueue(E e), boolean isEmpty(), E front(), E dequeue()`**
  Create a single **`Queue314`** object that stores **`Vertex`** objects: **`Queue314< Vertex>`**
- You may use the **`V get(Object key)`** and **`int size()`** methods from the **`Map`** interface.
- The **`Map vertices`** is not altered by this method.
- You may use the given **`Graph.clearAll`** method that sets all **`Vertex.scratch`** variables to 0.
- You may ***temporarily*** alter the **`Vertex`** objects connected by the edge in question by removing that edge, but must restore their adjacency lists before returning the result. The **`Vertex.removeEdge(Vertex dest)`** method may be used to do this.
- Use the **`Vertex.scratch`** variables as needed.
- You may use the following methods for **`java.util.List`**s.
  `int size(), E get(int pos), add(E val) /* adds to end*/, Iterator<E> iterator()`
- You may use the **`next`** and **`hasNext`** methods for **`Iterator`**s explicitly or implicitly.
- Do not create any new data structures other than the single instance of **`Queue314<Vertex>`** and **`Iterator`** objects.

Method to implement:

```
/* pre: v1 != null, v2 != null, !v1.equals(v2), The vertices specified
        by v1 and v2 are adjacent. There is an edge between them.
        This Graph is connected.
   post: return true if the specified edge is a bridge edge, false
       otherwise.
       This Graph is not altered after this method completes.
*/
public boolean isBridge(String v1, String v2)
```

Implement the method on the next page.

```
public boolean isBridge(String v1, String v2) {
```

EXAM # _____

3. **Trees.** (16 points) Write an instance method, **`numInRange`** for the **`IntBST`** class. The method returns the number of values in the binary search tree that are in range from the given **`low`** and **`high`** parameters **inclusive, [low, high]**. The **`IntBST`** class implements a binary search tree that only stores primitive **`int`**s.

Consider the following examples:

```
root of tree is 5
examples of calls to numInRange(int low, int high)

       5            numInRange(3, 6) -> returns 2
     /   \          numInRange(20, 30) -> returns 0
    3     12        numInRange(-5, -10) -> returns 0
   /     / \        numInRange(3, 3) -> returns 1
  0     9   15      numInRange(3, 5) -> returns 2
       /            numInRange(-5, 20) -> returns 7
      7             numInRange(0, 15) -> returns 7
                    numInRange(3, 10) -> returns 4
```

You may use the nested **`IntNode`** class.

```
public class IntBST {

    private IntNode root; // stores null if tree is empty
    // no size variable

    private static class IntNode {
        private int val;
        private IntNode left; // stores null if no left child
        private IntNode right; // stores null if no right child
```

You may not create any new data structures, <u>not even an array of length 1</u>.

If you create a helper method, do not include unnecessary parameters.

Do not add any class or instance variables to the `IntBST` class.

The calling object is not altered by this method.

Complete the `numInRange` instance method for the `IntBST` class on the next page.

```
/* pre: low <= high, post: per the problem description. */
public int numInRange(int low, int high) {
```

EXAM # _____

4. **Hash Tables** (17 points) Fully implement the **`HIterator`** class, an inner class of the **`HashTable314`** class. The **`HIterator`** class shall implement the **`Iterator`** interface including the **`hashNext`**, **`next`**, and **`remove`** methods. The purpose of a **`HIterator`** object is to allow clients to access all the elements of the **`HashTable314`**. (Recall, hash tables do not provide a **`E get(int pos)`** method.)

The **`HashTable314`** class uses closed addressing (chaining, buckets) to resolve collisions. The buckets in the internal array are objects that implement the **`java.util.List`** interface.

Elements in the **`HashTable314`** internal array that do not store any elements (empty buckets) store **`null`**.

Declare the necessary **`HIterator`** instance variables and implement a constructor if necessary.

Your implementation of **`hasNext`** shall be O(1) in all cases. The method is an accessor. The **`HIterator`** shall not be altered in **any way** when **`hasNext`** is called. Hint: Think about a primitive instance variable and what to initialize it to so that the **`hasNext`** check is a simple Boolean expression,

Do **NOT** check the preconditions for the **`next`** OR **`remove`** methods. Not required for this question.

If a bucket becomes empty due to the **`HIterator remove`** method, **`null`** out that element in the internal array.

You may create (via the **`iterator`** methods for **`List`**s) and use **`Iterator`** objects for the internal buckets as necessary. You may call the **`hasNext`**, **`next`**, and **`remove`** methods on those **`Iterator`**s.

Do **not** create any new data structures other than the **`Iterator`**s for internal buckets.

You may call the **`List.size()`** method on the non-**`null List`** objects stored in **`con`**.

Do **not** attempt to deal with potential **`ConcurrentModificationException`**s with the outer **`HashTable314`** object.

Do not use any other Java or methods or classes than those described above. Do add any methods or variables to the `HashTable314` class. Do not use recursion.

The **`HashTable314`** class:

```
public class HashTable314<E> implements Iterable<E> {

    private List<E>[] con;
    private int size; // number of elements in this HashTable314.

    public Iterator<E> iterator() { return new HIterator(); }

    private class HIterator implements Iterator<E> { // TO DO
```

```
private class HIterator implements Iterator<E> {
// Complete this class with instance variables, constructor if
// necessary, and the hasNext, next, and remove methods.

// More room on the last page if necessary.
```

If you need extra room for a coding question answer, place it on this page.
Make it clear which question you are answering, 2, 3, or 4.